for(i in 2:11){

if(i<10){

j<-paste("00",i,".csv",sep="")

print(j)

}

else if(i<100){

j<-paste("0",i,".csv",sep="")

print(j)

}

else{

j<-paste(i,".csv",sep="")

print(j)

}

}

**Attempt to extend to 332**

for(i in 4:332){

if(i<10){

j<-paste("00",i,".csv",sep="")

print(j)

}

else if(i<100){

j<-paste("0",i,".csv",sep="")

print(j)

}

else{

j<-paste(i,".csv",sep="")

print(j)

}

}

**Attempt to file.append**

for(i in 5:332){

if(i<10){

j<-paste("00",i,".csv",sep="")

file.append("mierda001.csv",j)

}

else if(i<100){

j<-paste("0",i,".csv",sep="")

file.append("mierda001.csv",j)

}

else{

j<-paste(i,".csv",sep="")

file.append("mierda001.csv",j)

}

}

**IT WORKS!**

Extract rows where id=11

ext1<-pol\_dat[pol\_dat["ID"]==11,]

**IT WORKS!**

sulf<-ext1[,"sulfate"]

good\_sulf<-complete.cases(sulf)

mean(good\_sulf)

[1] 0.2021908

> library("readxl")

> volumes<-read\_excel("To\_estimate\_line\_speeds.xls")

Error: `path` does not exist: ‘To\_estimate\_line\_speeds.xls’

> getwd()

[1] "C:/Users/javier.lores/Documents/Operations/Equipment"

> dir()

[1] "Line\_speeds.csv" "Speeds.csv"

[3] "To\_estimate\_line\_speeds.xlsx"

> volumes<-read\_excel("To\_estimate\_line\_speeds.xlsx")

> head(volumes)

# A tibble: 6 x 5

`Location Code` `Item No#` WeekC `Registering date`

<chr> <chr> <dttm> <dttm>

1 CALI FGCB50011 2017-12-31 00:00:00 2018-01-06 00:00:00

2 CALI FGCB50011 2018-02-11 00:00:00 2018-02-12 00:00:00

3 CALI FGCB50011 2018-02-18 00:00:00 2018-02-19 00:00:00

4 CALI FGCB50011 2018-02-18 00:00:00 2018-02-23 00:00:00

5 CALI FGCB50011 2018-03-04 00:00:00 2018-03-10 00:00:00

6 CALI FGCB50011 2018-03-25 00:00:00 2018-03-28 00:00:00

# ... with 1 more variable: `SumOfNet Weight` <dbl>

> head(volumes)

# A tibble: 6 x 5

`Location Code` `Item No#` WeekC `Registering date` `SumOfNet Weight`

<chr> <chr> <dttm> <dttm> <dbl>

1 CALI FGCB50011 2017-12-31 00:00:00 2018-01-06 00:00:00 9020

2 CALI FGCB50011 2018-02-11 00:00:00 2018-02-12 00:00:00 1880

3 CALI FGCB50011 2018-02-18 00:00:00 2018-02-19 00:00:00 1490

4 CALI FGCB50011 2018-02-18 00:00:00 2018-02-23 00:00:00 700

5 CALI FGCB50011 2018-03-04 00:00:00 2018-03-10 00:00:00 1720

6 CALI FGCB50011 2018-03-25 00:00:00 2018-03-28 00:00:00 1680

> mean(volumes[["SumOfNet Weight"]],na.rm=TRUE)

[1] 2477.513

> max(volumes[["SumOfNet Weight"]],na.rm=TRUE)

[1] 4545060

> min(volumes[["SumOfNet Weight"]],na.rm=TRUE)

[1] 1.125

>

**Programming assignment 1**

**Step by step…**

**Step 1**

pollutantmean<-function(directory) {

setwd("C:/Users/javier.lores/Documents/R")

d<-paste("C:/Users/javier.lores/Documents/R/",directory,sep="")

setwd(d)

file.copy("001.csv","pollution\_data.csv")

for(i in 2:332){

if(i<10){

j<-paste("00",i,".csv",sep="")

file.append("pollution\_data.csv",j)

}

else if(i<100){

j<-paste("0",i,".csv",sep="")

file.append("pollution\_data.csv",j)

}

else{

j<-paste(i,".csv",sep="")

file.append("pollution\_data.csv",j)

}

}

df<-read.csv("pollution\_data.csv")

file.remove("pollution\_data.csv")

}

Warning message:

In mean.default(pivot$Experimental) :

argument is not numeric or logical: returning NA

> d <- c("5","7")

> str(d)

chr [1:2] "5" "7"

> e <- as.numeric(d)

> str(e)

num [1:2] 5 7

> mean(d)

[1] NA

Warning message:

In mean.default(d) : argument is not numeric or logical: returning NA

> mean(e)

[1] 6

NO:

> mean(as.numeric(as.character(filas[["nitrate"]])),na.rm=TRUE)

[1] 1572.815

**YES:**

**mean(as.numeric(as.character(filas[["nitrate"]])),na.rm=TRUE)**

**[1] 0.6530321**

**Step 2**

pollutantmean<-function(directory,pollutant="nitrate") {

setwd("C:/Users/javier.lores/Documents/R")

d<-paste("C:/Users/javier.lores/Documents/R/",directory,sep="")

setwd(d)

file.copy("001.csv","pollution\_data.csv")

for(i in 2:332){

if(i<10){

j<-paste("00",i,".csv",sep="")

file.append("pollution\_data.csv",j)

}

else if(i<100){

j<-paste("0",i,".csv",sep="")

file.append("pollution\_data.csv",j)

}

else{

j<-paste(i,".csv",sep="")

file.append("pollution\_data.csv",j)

}

}

df<-read.csv("pollution\_data.csv")

file.remove("pollution\_data.csv")

m\_df<-mean(as.numeric(as.character(df[[pollutant]])),na.rm=TRUE)

m\_df

}

**Algo es algo…**

> pollutantmean("specdata")

[1] 1.702932

Warning message:

In mean(as.numeric(as.character(df[[pollutant]])), na.rm = TRUE) :

NAs introduced by coercion

>pollutantmean("specdata","sulfate")

[1] 3.189369

Warning message:

In mean(as.numeric(as.character(df[[pollutant]])), na.rm = TRUE) :

NAs introduced by coercion

**Step 3**

pollutantmean<-function(directory="specdata",pollutant="nitrate",ID=1:332) {

setwd("C:/Users/javier.lores/Documents/R")

d<-paste("C:/Users/javier.lores/Documents/R/",directory,sep="")

setwd(d)

file.copy("001.csv","pollution\_data.csv")

for(i in 2:332){

if(i<10){

j<-paste("00",i,".csv",sep="")

file.append("pollution\_data.csv",j)

}

else if(i<100){

j<-paste("0",i,".csv",sep="")

file.append("pollution\_data.csv",j)

}

else{

j<-paste(i,".csv",sep="")

file.append("pollution\_data.csv",j)

}

}

df<-read.csv("pollution\_data.csv")

file.remove("pollution\_data.csv")

df\_ID<-df[df["ID"]==ID,]

Mdf\_ID<-mean(as.numeric(as.character(df\_ID[[pollutant]])),na.rm=TRUE)

Mdf\_ID

}

Conclusion: df\_ID aparece con filas de menos, debe ser por ID character al hacer file.append versus 1:322 as number

Classes of all columns in a dataframe

> foo <- data.frame(c("a", "b"), c(1, 2))

> names(foo) <- c("SomeFactor", "SomeNumeric")

> lapply(foo, class)

$SomeFactor

[1] "factor"

$SomeNumeric

[1] "numeric"

cols = c(1, 3, 4, 5);

df[,cols] = apply(df[,cols], 2, function(x) as.numeric(as.character(x)));

Attempt… and it works

Assignment 1

Part 1

**pollutantmean<-function(directory="specdata",pollutant="nitrate",Ident=1:332) {**

**file.remove("pollution\_data.csv")**

**file.remove("ddff.csv")**

**file.remove("df\_ID.csv")**

**file.remove("Mdf\_ID.csv")**

**setwd("C:/Users/javier.lores/Documents/R")**

**d<-paste("C:/Users/javier.lores/Documents/R/",directory,sep="")**

**setwd(d)**

**file.copy("001.csv","pollution\_data.csv")**

**for(i in 2:332){**

**if(i<10){**

**j<-paste("00",i,".csv",sep="")**

**file.append("pollution\_data.csv",j)**

**}**

**else if(i<100){**

**j<-paste("0",i,".csv",sep="")**

**file.append("pollution\_data.csv",j)**

**}**

**else{**

**j<-paste(i,".csv",sep="")**

**file.append("pollution\_data.csv",j)**

**}**

**}**

**ddff<-read.csv("pollution\_data.csv")**

**ddff[,4]<-as.numeric(as.character(ddff[,4]))**

**df\_ID<-subset(ddff,ID>=min(Ident) & ID<=max(Ident))**

**write.csv(ddff,"ddff.csv")**

**write.csv(df\_ID,"df\_ID.csv")**

**Mdf\_ID<-mean(as.numeric(as.character(df\_ID[[pollutant]])),na.rm=TRUE)**

**write.csv(Mdf\_ID,"Mdf\_ID.csv")**

**Mdf\_ID**

**}**

Done…

df\_ID<-subset(ddff,ID==Ident | ID==Ident+1)

df\_ID<-ddff[ddff["ID"]==ID,]

file.remove("pollution\_data.csv")

Mdf\_ID<-mean(as.numeric(as.character(df\_ID[[pollutant]])),na.rm=TRUE)

Mdf\_ID

media\_calc<-function(uno,dos,tres){

u<-3

d<-dos

tt<-tres

mn<-mean(c(uno\*u,dos,tres),na.rm=TRUE)

}

U, d, tt, mn do not exist once the function has finished the operation

Change class of a column in a dataframe

for (i in 2:3{

mydf[,i] <- as.factor(mydf[,i])

}

Assignment 1

PART 2

**complete<-function(directory="specdata",Ident=1:332) {**

**file.remove("pollution\_data.csv")**

**file.remove("com\_ddff.csv")**

**setwd("C:/Users/javier.lores/Documents/R")**

**d<-paste("C:/Users/javier.lores/Documents/R/",directory,sep="")**

**setwd(d)**

**file.copy("001.csv","pollution\_data.csv")**

**for(i in 2:332){**

**if(i<10){**

**j<-paste("00",i,".csv",sep="")**

**file.append("pollution\_data.csv",j)**

**}**

**else if(i<100){**

**j<-paste("0",i,".csv",sep="")**

**file.append("pollution\_data.csv",j)**

**}**

**else{**

**j<-paste(i,".csv",sep="")**

**file.append("pollution\_data.csv",j)**

**}**

**}**

**ddff<-read.csv("pollution\_data.csv")**

**ddff[,4]<-as.numeric(as.character(ddff[,4]))**

**write.csv(ddff,"com\_ddff.csv")**

**x<-numeric(length(Ident))**

**y<-numeric(length(Ident))**

**for(k in 1:length(Ident)){**

**x[k]<-sum(complete.cases(subset(ddff,ID==Ident[k])))**

**y[k]<-Ident[k]**

**}**

**comp<-data.frame(y, x, stringsAsFactors=FALSE)**

**names(comp)<-c("ID","nobs")**

**print(comp)**

**}**

Fails for question 7

set.seed(42)

cc <- complete("specdata", 332:1)

use <- sample(332, 10)

print(cc[use, "nobs"])

To resolve issue on question 7… **IT WORKS!**

**complete<-function(directory="specdata",Ident=1:332) {**

**file.remove("pollution\_data.csv")**

**file.remove("com\_ddff.csv")**

**setwd("C:/Users/javier.lores/Documents/R")**

**d<-paste("C:/Users/javier.lores/Documents/R/",directory,sep="")**

**setwd(d)**

**file.copy("001.csv","pollution\_data.csv")**

**for(i in 2:332){**

**if(i<10){**

**j<-paste("00",i,".csv",sep="")**

**file.append("pollution\_data.csv",j)**

**}**

**else if(i<100){**

**j<-paste("0",i,".csv",sep="")**

**file.append("pollution\_data.csv",j)**

**}**

**else{**

**j<-paste(i,".csv",sep="")**

**file.append("pollution\_data.csv",j)**

**}**

**}**

**ddff<-read.csv("pollution\_data.csv")**

**ddff[,4]<-as.numeric(as.character(ddff[,4]))**

**write.csv(ddff,"com\_ddff.csv")**

**x<-numeric(length(Ident))**

**y<-numeric(length(Ident))**

**for(k in Ident[1]:Ident[length(Ident)]){**

**x[k]<-sum(complete.cases(subset(ddff,ID==Ident[k])))**

**y[k]<-Ident[k]**

**}**

**comp<-data.frame(y, x, stringsAsFactors=FALSE)**

**names(comp)<-c("ID","nobs")**

**print(comp)**

**}**

Attempts, failures, ideas…

comp= data.frame(x = numeric(), y = numeric())

for(k in length(Ident)){

x[k]<-sum(complete.cases(subset(ddff,ID==k)))

y[k]<-k

}

comp<-data.frame(y, x, stringsAsFactors=FALSE)

INFO

Loop with the content of a variable, even in reverse order

prueba<-function(h){

for(i in h)

print(i)

}

> prueba(3:1)

[1] 3

[1] 2

[1] 1

>

f4 <- function(n) {

x <- numeric(n)

y <- character(n)

for (i in 1:n) {

x[i] <- i

y[i] <- i

}

data.frame(x, y, stringsAsFactors=FALSE)

}

n<-sum(complete.cases(subset(df,ID<=min(Ident) & ID>=max(Ident))))

Esto CUENTA los TRUEs de la lista de TRUEs y FALSEs

El total:

> sum(comp$nobs)

[1] 111802

PART 3

**corr<-function(directory="specdata",threshold=0){**

**Ident<-1:332**

**file.remove("pollution\_data.csv")**

**file.remove("ddff.csv")**

**file.remove("ddff\_comps.csv")**

**file.remove("correla.csv")**

**file.remove("for\_cor.csv")**

**file.remove("ks.csv")**

**setwd("C:/Users/javier.lores/Documents/R")**

**d<-paste("C:/Users/javier.lores/Documents/R/",directory,sep="")**

**setwd(d)**

**file.copy("001.csv","pollution\_data.csv")**

**for(i in 2:332){**

**if(i<10){**

**j<-paste("00",i,".csv",sep="")**

**file.append("pollution\_data.csv",j)**

**}**

**else if(i<100){**

**j<-paste("0",i,".csv",sep="")**

**file.append("pollution\_data.csv",j)**

**}**

**else{**

**j<-paste(i,".csv",sep="")**

**file.append("pollution\_data.csv",j)**

**}**

**}**

**ddff<-read.csv("pollution\_data.csv")**

**ddff[,2]<-as.numeric(as.character(ddff[,2]))**

**ddff[,3]<-as.numeric(as.character(ddff[,3]))**

**ddff[,4]<-as.numeric(as.character(ddff[,4]))**

**write.csv(ddff,"ddff.csv")**

**x<-numeric(length(Ident))**

**y<-numeric(length(Ident))**

**for(k in 1:length(Ident)){**

**x[k]<-sum(complete.cases(subset(ddff,ID==Ident[k])))**

**y[k]<-Ident[k]**

**}**

**comp<-data.frame(y, x, stringsAsFactors=FALSE)**

**names(comp)<-c("ID","nobs")**

**write.csv(comp,"comp\_ddff.csv")**

**as.character(t)**

**as.numeric(u)**

**as.numeric(v)**

**as.numeric(w)**

**m<-numeric()**

**m<-0**

**for(k in 1:nrow(ddff)){**

**if(is.na(ddff[k,2])==FALSE){**

**if(is.na(ddff[k,3])==FALSE){**

**m<-m+1**

**t[m]<-ddff[k,1]**

**u[m]<- ddff[k,2]**

**v[m] <- ddff[k,3]**

**w[m] <- ddff[k,4]**

**}**

**}**

**}**

**ddff\_comps<-data.frame(t, u, v, w, stringsAsFactors=FALSE)**

**names(ddff\_comps)<-c("Date","sulfate","nitrate","ID")**

**write.csv(ddff\_comps,"ddff\_comps.csv")**

**nodo<-numeric()**

**correla<-numeric()**

**q<-0**

**h<-0**

**for(k in 1:nrow(comp)){**

**if(comp[k,2]>=threshold){**

**cor\_s<-numeric(comp[k,2])**

**cor\_n<-numeric(comp[k,2])**

**for(o in 1:comp[k,2]){**

**cor\_s[o]<-ddff\_comps[o+h,2]**

**cor\_n[o]<-ddff\_comps[o+h,3]**

**}**

**h<-h+comp[k,2]**

**for\_cor<- matrix(c(cor\_s, cor\_n), nrow=length(cor\_s),ncol=2)**

**nodo[k]<-k**

**rm(cor\_s)**

**rm(cor\_n)**

**q<-q+1**

**write.csv(for\_cor,"for\_cor.csv")**

**correla[q]<-cor(for\_cor[,1],for\_cor[,2])**

**}**

**else{**

**h<-h+comp[k,2]}**

**}**

**write.csv(nodo,"ks.csv")**

**write.csv(correla,"correla.csv")**

**correla**

**}**

Pregunta 8

cr <- corr("specdata")

cr <- sort(cr)

set.seed(868)

out <- round(cr[sample(length(cr), 5)], 4)

print(out)

***> cr<-sort(cr)***

***> set.seed(868)***

***> out <- round(cr[sample(length(cr), 5)], 4)***

***> print(out)***

***[1] 0.2688 0.1127 -0.0085 0.4586 0.0447***

Pregunta 9

cr <- corr("specdata", 129)

cr <- sort(cr)

n <- length(cr)

set.seed(197)

out <- c(n, round(cr[sample(n, 5)], 4))

print(out)

***> cr <- corr("specdata", 129)***

***Warning messages:***

***1: In corr("specdata", 129) : NAs introduced by coercion***

***2: In corr("specdata", 129) : NAs introduced by coercion***

***3: In corr("specdata", 129) : NAs introduced by coercion***

***> cr <- sort(cr)***

***> n <- length(cr)***

***> set.seed(197)***

***> out <- c(n, round(cr[sample(n, 5)], 4))***

***> print(out)***

***[1] 243.0000 0.2540 0.0504 -0.1462 -0.1680 0.5969***

Ddff\_comps\_thr<-ddff[ddff[,ID>=threshold]

if(x[k]>=threshold){

vec[k]<-comp$ID

}

#laply para cor to new dataframe

z[k]<-y[k]

(subset(ddff,ID==y[k]))

ddffc<-subset(ddff,ddff[,2][!is.na(ddff[,2])] & ddff[,3][!is.na(ddff[,3])])

x[k]<-sum(complete.cases(subset(ddff,ID==Ident[k])))

y[k]<-Ident[k]

}

}

}

prueba<-function(x){

if(exampl$Date!=2003-01-01){

h<-1

print(h)

}

}

Remove zero 0

down vote[favorite](https://stackoverflow.com/questions/29639680/r-table-function-how-to-remove-0-counts)

1

I need to remove the rows from the table function output, which have 0 counts in all the columns. Is there any easy way to do that?

table(ds$animal,ds$gender)

\_\_\_ | M | F

Cat | 9 | 4

Dog | 0 | 0

Rat | 4 | 3

I just would like to see those rows:

\_\_\_ | M | F

Cat | 9 | 4

Rat | 4 | 3

Complete.cases

I'd like to remove the lines in this data frame that:

a) **contain NAs across all columns.** Below is my example data frame.

gene hsap mmul mmus rnor cfam

1 ENSG00000208234 0 NA NA NA NA

2 ENSG00000199674 0 2 2 2 2

3 ENSG00000221622 0 NA NA NA NA

4 ENSG00000207604 0 NA NA 1 2

5 ENSG00000207431 0 NA NA NA NA

6 ENSG00000221312 0 1 2 3 2

Basically, I'd like to get a data frame such as the following.

gene hsap mmul mmus rnor cfam

2 ENSG00000199674 0 2 2 2 2

6 ENSG00000221312 0 1 2 3 2

b) **contain NAs in only some columns**, so I can also get this result:

gene hsap mmul mmus rnor cfam

2 ENSG00000199674 0 2 2 2 2

4 ENSG00000207604 0 NA NA 1 2

6 ENSG00000221312 0 1 2 3 2

[r](https://stackoverflow.com/questions/tagged/r) [dataframe](https://stackoverflow.com/questions/tagged/dataframe) [filter](https://stackoverflow.com/questions/tagged/filter) [missing-data](https://stackoverflow.com/questions/tagged/missing-data) [r-faq](https://stackoverflow.com/questions/tagged/r-faq)

844down voteaccepted

Also check [complete.cases](http://stat.ethz.ch/R-manual/R-patched/library/stats/html/complete.cases.html) :

> final[complete.cases(final), ]

gene hsap mmul mmus rnor cfam

2 ENSG00000199674 0 2 2 2 2

6 ENSG00000221312 0 1 2 3 2

na.omit is nicer for just removing all NA's. complete.cases allows partial selection by including only certain columns of the dataframe:

> final[complete.cases(final[ , 5:6]),]

gene hsap mmul mmus rnor cfam

2 ENSG00000199674 0 2 2 2 2

4 ENSG00000207604 0 NA NA 1 2

6 ENSG00000221312 0 1 2 3 2

Your solution can't work. If you insist on using is.na, then you have to do something like:

> final[rowSums(is.na(final[ , 5:6])) == 0, ]

gene hsap mmul mmus rnor cfam

2 ENSG00000199674 0 2 2 2 2

4 ENSG00000207604 0 NA NA 1 2

6 ENSG00000221312 0 1 2 3 2

but using complete.cases is quite a lot more clear, and faster.

8down voteaccepted

you need to drop levels from the factor animal.

table(droplevels(ds$animal),ds$gender)

you can also just drop them from ds and then do the table

ds$anima <- droplevels(ds$animal)

with(ds, table(animal,gender))

here I used with because it prints headers.

regex

One of the most frequently used string recognition algorithms out there is regex and R implements regex.  However, users can often be frustrated with how despite taking examples verbatim from many sources such as stackoverflow they do not seem to work.  From my own experience, I have found that the largest issue is really about what characters need to be escaped from R.

For example:

**Listing all files whose names match a simple pattern.**

Looking at “/^.\*icon.\*\.png$/i” from

[http://stackoverflow.com/questions/4845125/regex-to-match-filename-containing-a-word-regardless-of-case](https://stackoverflow.com/questions/4845125/regex-to-match-filename-containing-a-word-regardless-of-case)

I was able to get “^.\*icon.\*.png$“ to work in R though I lost the case insensitivity.  I think including the “^.” ensures that only files in the current directory, not subdirectory are matched but I am not sure.

So, the following code will return a list of file names from the folder Clipart which match the pattern [anything]icon.png

list.files(“C:/Clipart/”, pattern=”^.\*icon.\*.png$”)

[1] “manicon.png”     “handicon.png”     “bookicon.png”

Looking at the original entry we can see that what was causing us problems was the attempt to escape the “^” which does not need to be escaped in R.

Before looking at another example lets modify the previous command slightly to show how we can make it match differently.

list.files(“C:/Clipart/”, pattern=”^.\*icon\*.\*.png$”)

[1] “manicon.png”     “handicon.png”     “bookicon.png”    “iconnew.png”

There are a lot of resources available for regex since it is really its own text matching language supported by many different programming languages.  A good introductory guide can be found:

<http://www.zytrax.com/tech/web/regex.htm>

or

[http://www.regular-expressions.info/tutorial.html](https://www.regular-expressions.info/tutorial.html)

> list.files("C:/Users/javier.lores/Documents/R/color", pattern="\*olor\*.\*.xlsx$")

[1] "Color1.xlsx" "color10 lots 2016-2023.xlsx" "Color11 Medium 5 belly.xlsx" "Color12 Regular Medium 6.xlsx"

[5] "Color13 Regular Medium 7.xlsx" "Color14 Regular Small 2.xlsx" "Color15 Regular Small 3.xlsx" "Color16 Large Golden Flake.xlsx"

[9] "Color2.xlsx" "Color3 Cracklin Medium Tender.xlsx" "Color4 Cracklin Short Strips.xlsx" "Color5 Cracklin Small Tender.xlsx"

[13] "Color6 Cracklin Strips.xlsx" "Color7 Gaytan Small 2.xlsx" "Color8 Ham Strips.xlsx" "color9 lots 1x9.xlsx"

28down vote

glob2rx() converts a pattern including a wildcard into the equivalent regular expression. You then need to pass this regular expression onto one of R's pattern matching tools.

If you want to match "blue\*" where \* has the usual wildcard, *not* regular expression, meaning we use glob2rx() to convert the wildcard pattern into a useful regular expression:

> glob2rx("blue\*")

[1] "^blue"

The returned object *is* a regular expression.

Given your data:

x <- c('red','blue1','blue2', 'red2')

we can pattern match using grep() or similar tools:

> grx <- glob2rx("blue\*")

> grep(grx, x)

[1] 2 3

> grep(grx, x, value = TRUE)

[1] "blue1" "blue2"

> grepl(grx, x)

[1] FALSE TRUE TRUE FALSE

As for the selecting rows problem you posted

> a <- data.frame(x = c('red','blue1','blue2', 'red2'))

> with(a, a[grepl(grx, x), ])

[1] blue1 blue2

Levels: blue1 blue2 red red2

> with(a, a[grep(grx, x), ])

[1] blue1 blue2

Levels: blue1 blue2 red red2

or via subset():

> with(a, subset(a, subset = grepl(grx, x)))

x

2 blue1

3 blue2

Color data analysis

> library(readxl)

> color\_data<-read\_xlsx("Color.xlsx")

> write.csv(color\_data,"color.csv")

closeAllConnections()

Un solo fichero de color…

**colo<-function(x=0){**

**archivos<-dir()**

**colr<-read\_xlsx(archivos[1])**

**write.csv(colr,"color\_d.csv")**

**rm(colr)**

**for(i in 2:length(archivos)){**

**colr<-read\_xlsx(archivos[i])**

**write.csv(colr,"colr.csv")**

**file.append("color\_d.csv","colr.csv")**

**rm(colr)**

**file.remove("colr.csv")**

**}**

**color\_d<-read.csv("color\_d.csv")**

**head(color\_d)**

**}**

write.csv(color\_data,"color\_data.csv")

length(colores)

}

hist(B, col = "red", breaks=6, xlim=c(0,max),   
main="My Histogram", las=2, xlab = "Values", cex.lab = 1.3)

Select files by pattern  
2down voteaccepted

You can list the files in your working directory with list.files(). It has an argument allowing you to specify a certain pattern:

list.files(pattern = "Alta Guajira")

# [1] "Alta Guajira Coord & elevation TS 0-4000.txt"

If you have a vector with the patterns in there, like:

patterns <- c("Alta Guajira", "Rio Tomo", "Rio Sogamoso")

sapply(patterns, function(x){list.files(pattern = x)})

Edit

If you already have the file names, you are basically looking for patterns in a character vector.

df <- c("Alta Guajira Coord & elevation TS 0-4000.txt","Baja Guajira Coord & elevation TS 0-4000.txt",

"Bajo Meta Coord & elevation TS 0-4000.txt", "Rio Arauca Coord & elevation TS 0-4000.txt",

"Rio Catatubo Coord & elevation TS 0-4000.txt", "Rio Cesar Coord & elevation TS 0-4000.txt",

"Rio Sogamoso Coord & elevation TS 0-4000.txt", "Rio Tomo Coord & elevation TS 0-4000.txt",

"Sabana de Bogota Coord & elevation TS 0-4000.txt", "Total Area Coord & elevation TS 0-4000.txt")

patterns <- c("Alta Guajira", "Rio Tomo", "Rio Sogamoso")

it <- 0

res <- c()

for(i in patterns){

it <- it + 1

res <- append(res, df[grepl(pattern = i, x = df)])

}

res

# [1] "Alta Guajira Coord & elevation TS 0-4000.txt" "Rio Tomo Coord & elevation TS 0-4000.txt"

# [3] "Rio Sogamoso Coord & elevation TS 0-4000.txt"

Or, alternatively:

res2 <- sapply(patterns, function(y){df[grepl(pattern = y, x = df)]})

lista<-list.files(pattern="olor")

class(lista)

[1] "character"

colr<-vector()

for (i in 1:length(lista)){

colr[i]<-read\_xlsx(lista[1])

}

lapply(

split and lapply



printmessage<-function(x){

if(x>0)

print("x is greater than zero")

else

print("x is equal to or less than zero")

invisible(x)

}

Digo yo: SET the SEED so that every time you run the random number you get the same result

lapply(unique\_vals,function(elem)elem[2])

anonymous function, created on the fly…

we can

| use vapply(flags, class, character(1)). The 'character(1)' argument tells R

| that we expect the class function to return a **character vector of length 1**

| when applied to EACH column of the flags dataset.

> table(flags$landmass)

1 2 3 4 5 6

31 17 35 52 39 20

> table(flags$animate)

0 1

155 39

Week 3

Quiz

> tapply(iris$Sepal.Length,iris$Species=="virginica",mean)

FALSE TRUE

5.471 6.588

> f<-gl(3,50)

> tapply(iris$Sepal.Length,f,mean)

1 2 3

5.006 5.936 6.588

> apply(iris[, 1:4], 2, mean)

Sepal.Length Sepal.Width Petal.Length Petal.Width

5.843333 3.057333 3.758000 1.199333

How can one calculate the average miles per gallon (mpg) by number of cylinders in the car (cyl)? Select all that apply.



apply(mtcars, 2, mean)

**Un-selected is correct**



sapply(split(mtcars$mpg, mtcars$cyl), mean)

> tapply(mtcars$mpg,mtcars$cyl,mean)

4 6 8

26.66364 19.74286 15.10000

> hps<-tapply(mtcars$hp,mtcars$cyl,mean)

> hps[3]-hps[1]

8

126.5779

> tapply(mtcars$hp,mtcars$cyl,mean)

4 6 8

82.63636 122.28571 209.21429

Whole quiz

Take a look at the 'iris' dataset that comes with R. The data can be loaded with the code:



1

2

library(datasets)

data(iris)

A description of the dataset can be found by running



1

?iris

There will be an object called 'iris' in your workspace. In this dataset, what is the mean of 'Sepal.Length' for the species virginica? **Please round your answer to the nearest whole number**.

(Only enter the numeric result and nothing else.)



**Correct Response**

To get the answer here, you can use 'tapply' to calculate the mean of 'Sepal.Length' within each species.

Question 2

Correct

1 / 1 points

## 2. Question 2

Continuing with the 'iris' dataset from the previous Question, what R code returns a vector of the means of the variables 'Sepal.Length', 'Sepal.Width', 'Petal.Length', and 'Petal.Width'?



rowMeans(iris[, 1:4])



colMeans(iris)



apply(iris, 2, mean)



apply(iris, 1, mean)



apply(iris[, 1:4], 2, mean)

**Correct**



apply(iris[, 1:4], 1, mean)

Question 3

Correct

1 / 1 points

## 3. Question 3

Load the 'mtcars' dataset in R with the following code



1

2

library(datasets)

data(mtcars)

There will be an object names 'mtcars' in your workspace. You can find some information about the dataset by running



1

?mtcars

How can one calculate the average miles per gallon (mpg) by number of cylinders in the car (cyl)? Select all that apply.



split(mtcars, mtcars$cyl)

**Un-selected is correct**



tapply(mtcars$mpg, mtcars$cyl, mean)

**Correct**



apply(mtcars, 2, mean)

**Un-selected is correct**



sapply(split(mtcars$mpg, mtcars$cyl), mean)

**Correct**



tapply(mtcars$cyl, mtcars$mpg, mean)

**Un-selected is correct**



mean(mtcars$mpg, mtcars$cyl)

**Un-selected is correct**



lapply(mtcars, mean)

**Un-selected is correct**



sapply(mtcars, cyl, mean)

**Un-selected is correct**



with(mtcars, tapply(mpg, cyl, mean))

**Correct**

Correct

1 / 1 points

## 4. Question 4

Continuing with the 'mtcars' dataset from the previous Question, what is the absolute difference between the average horsepower of 4-cylinder cars and the average horsepower of 8-cylinder cars?

(**Please round your final answer to the nearest whole number**. Only enter the numeric result and nothing else.)



**Correct Response**

Question 5

Correct

1 / 1 points

## 5. Question 5

If you run



1

debug(ls)

what happens when you next call the 'ls' function?



The 'ls' function will return an error.



Execution of 'ls' will suspend at the beginning of the function and you will be in the browser.

**Correct**



The 'ls' function will execute as usual.



You will be prompted to specify at which line of the function you would like to suspend execution and enter the browser.

Week 3

Programming assignment 2

Example provided:

makeVector <- function(x = numeric()) {

m <- NULL

set <- function(y) {

x <<- y

m <<- NULL

}

get <- function() x

setmean <- function(mean) m <<- mean

getmean <- function() m

list(set = set, get = get,

setmean = setmean,

getmean = getmean)

}

cachemean <- function(x, ...) {

m <- x$getmean()

if(!is.null(m)) {

message("getting cached data")

return(m)

}

data <- x$get()

m <- mean(data, ...)

x$setmean(m)

m

}

**Pruebesitas**

makeVector <- function(x = numeric()) {

m <- NULL

set <- function(y) {

x <<- y

m <<- NULL

}

get <- function() x

setmean <- function(mean) m <<- mean

getmean <- function() m

y<-list(set = set, get = get,

setmean = setmean,

getmean = getmean)

print(y)

}

**Otra pruebesita** ->> NO funciona

cachemean <- function(x, ...) {

makeVector <- function(x = numeric()) {

m <- NULL

set <- function(y) {

x <<- y

m <<- NULL

}

get <- function() x

setmean <- function(mean) m <<- mean

getmean <- function() m

list(set = set, get = get,

setmean = setmean,

getmean = getmean)

}

m <- x$getmean()

if(!is.null(m)) {

message("getting cached data")

return(m)

}

data <- x$get()

m <- mean(data, ...)

x$setmean(m)

m

}

**Otra pruebesita**:

Cachemean (makeVector(y))

write.csv(x,file="valor\_x.csv") después de m<-x$getmean

Error in as.data.frame.default(x[[i]], optional = TRUE) :

cannot coerce class ‘"function"’ to a data.frame

**Otra pruebesita**

makeVector <- function(x = numeric()) {

set <- function(y) {

x <<- y

}

get <- function() x

setmean <- function(mean) m <<- mean

getmean <- function() m

list(set = set, get = get,

setmean = setmean,

getmean = getmean)

}

> cachemean(makeVector(y))

[1] 7 8 9

Error: $ operator is invalid for atomic vectors

> r=matrix()

> r

[,1]

[1,] NA

> m\_inv<-NULL

> m<-mean

> m

function (x, ...)

UseMethod("mean")

<bytecode: 0x000000001a2e1df8>

<environment: namespace:base>

> m(3)

[1] 3

> m(c(3,5))

[1] 4

> m<-solve

> m(matrix(c(1,2,3,4),ncol=2,nrow=2))

[,1] [,2]

[1,] -2 1.5

Attempt

makeCacheMatrix <- function(x = matrix()) {

inv <- NULL

set <- function(y) {

x <<- y

inv <<- NULL

}

get <- function() x

setsolve <- function(solve) inv <<- solve

getsolve <- function() inv

list(set = set, get = get,

setsolve = setsolve,

getsolve = getsolve)

}

cacheSolve <- function(x, ...) {

inv <- x$getsolve()

if(!is.null(inv)) {

message("getting cached data")

return(inv)

}

else {

data <- x$get()

inv <- solve(data, ...)

x$setsolve(inv)

inv}

}

He quitado esto…

else if(x==y) {

message("getting cached data")

return(inv)

}

Week 4

Quiz

**Congratulations! You passed!**

Next Item

Question 1

Correct

1 / 1 points

## 1. Question 1

What is produced at the end of this snippet of R code?



1

2

set.seed(1)

rpois(5, 2)



It is impossible to tell because the result is random



A vector with the numbers 1, 1, 2, 4, 1

**Correct**

Because the `set.seed()' function is used, `rpois()' will always output the same vector in this code.



A vector with the numbers 1, 4, 1, 1, 5



A vector with the numbers 3.3, 2.5, 0.5, 1.1, 1.7

Question 2

Correct

1 / 1 points

## 2. Question 2

What R function can be used to generate standard Normal random variables?



dnorm



pnorm



rnorm

**Correct**

Functions beginning with the `r' prefix are used to simulate random variates.



qnorm

Question 3

Correct

1 / 1 points

## 3. Question 3

When simulating data, why is using the set.seed() function important? Select all that apply.



It ensures that the random numbers generated are within specified boundaries.

**Un-selected is correct**



It can be used to generate non-uniform random numbers.

**Un-selected is correct**



It can be used to specify which random number generating algorithm R should use, ensuring consistency and reproducibility.

**Correct**



It ensures that the sequence of random numbers is truly random.

**Un-selected is correct**

Question 4

Correct

1 / 1 points

## 4. Question 4

Which function can be used to evaluate the inverse cumulative distribution function for the Poisson distribution?



rpois



dpois



ppois



qpois

**Correct**

Probability distribution functions beginning with the `q' prefix are used to evaluate the quantile (inverse cumulative distribution) function.

Question 5

Correct

1 / 1 points

## 5. Question 5

What does the following code do?



1

2

3

4

set.seed(10)

x <- rep(0:1, each = 5)

e <- rnorm(10, 0, 20)

y <- 0.5 + 2 \* x + e



Generate uniformly distributed random data



Generate random exponentially distributed data



Generate data from a Poisson generalized linear model



Generate data from a Normal linear model

**Correct**

Question 6

Correct

1 / 1 points

## 6. Question 6

What R function can be used to generate Binomial random variables?



pbinom



rbinom

**Correct**



qbinom



dbinom

Question 7

Correct

1 / 1 points

## 7. Question 7

What aspect of the R runtime does the profiler keep track of when an R expression is evaluated?



the global environment



the function call stack

**Correct**



the working directory



the package search list

Question 8

Correct

1 / 1 points

## 8. Question 8

Consider the following R code



1

2

3

4

library(datasets)

Rprof()

fit <- lm(y ~ x1 + x2)

Rprof(NULL)

(Assume that y, x1, and x2 are present in the workspace.) Without running the code, what percentage of the run time is spent in the 'lm' function, based on the 'by.total' method of normalization shown in 'summaryRprof()'?



100%

**Correct**

When using `by.total' normalization, the top-level function (in this case, `lm()') always takes 100% of the time.



It is not possible to tell



23%



50%

Question 9

Correct

1 / 1 points

## 9. Question 9

When using 'system.time()', what is the user time?



It is the time spent by the CPU evaluating an expression

**Correct**



It is the time spent by the CPU waiting for other tasks to finish



It is a measure of network latency



It is the "wall-clock" time it takes to evaluate an expression

Question 10

Incorrect

0 / 1 points

## 10. Question 10

If a computer has more than one available processor and R is able to take advantage of that, then which of the following is true when using 'system.time()'?



user time is 0



elapsed time may be smaller than user time



elapsed time is 0



user time is always smaller than elapsed time

**This should not be selected**

I think it is the second. I need to re-watch the video.

type ls() to see a list of the variables in your workspace. Then, type

| rm(list=ls()) to clear your workspace.

| Whenever you're working with a new dataset, the first thing you should do is

| look at it! What is the format of the data? What are the dimensions? What are

| the variable names? How are the variables stored? Are there missing data? Are

| there any flaws in the data?

| Each of the functions we've introduced so far has its place in helping you to

| better understand the structure of your data. However, we've left the best

| for last....

Let's simulate rolling four six-sided dice: sample(1:6, 4, replace = TRUE)

| You are really on a roll!

| Now use replicate(100, rpois(5, 10)) to perform this operation 100 times.

| Store the result in a new variable called my\_pois.

| replicate() created a matrix, each column of which contains 5 random numbers

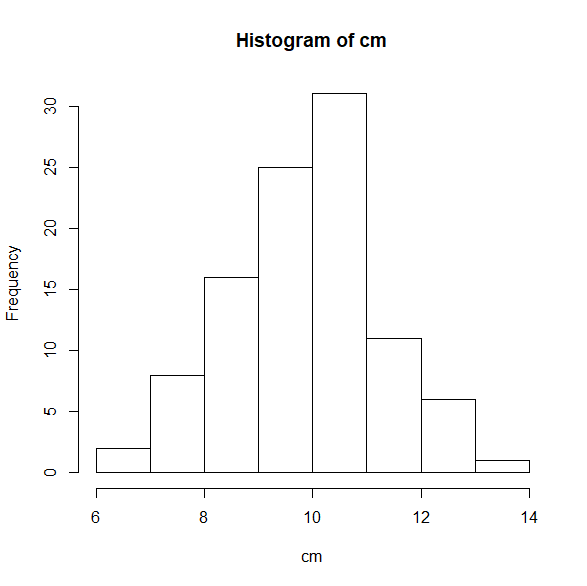
| generated from a Poisson distribution with mean 10. Now we can find the mean

| of each column in my\_pois using the colMeans() function. Store the result in

| a variable called cm.

| And let's take a look at the distribution of our column means by plotting a

| histogram with hist(cm).



| Looks like our column means are almost normally distributed, right? That's

| the Central Limit Theorem at work, but that's a lesson for another day!

| Simulation is practically a field of its own and we've only skimmed the

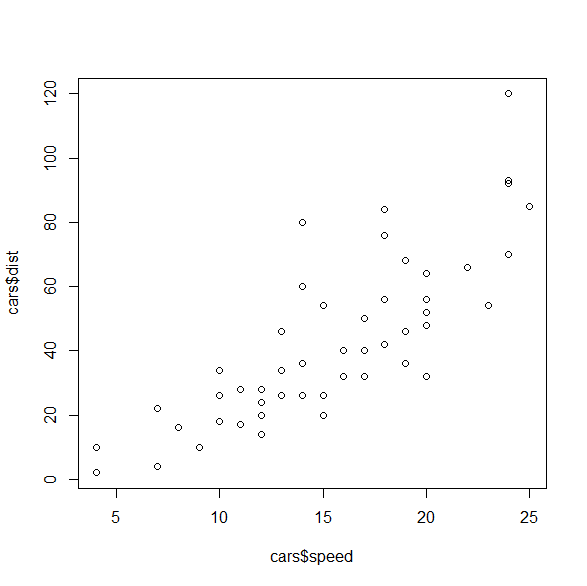
| surface of what's possible. I encourage you to explore these and other

| functions further on your own.

| These include lattice, ggplot2 and ggvis.

data(cars)

plot(x = cars$speed, y = cars$dist).



Check Value included in vector

> "NY" %in% outcome[,7]

[1] TRUE

grep("ttack",names(outcome))

[1] 11 12 13 14 15 16 29 30 31 32 33 34

install.packages("stringr")

condition<-c("heart attack")

> word(condition,1,2)

[1] "heart attack"

> word(condition,1,1)

[1] "heart"

Stack overflow  
34down vote

I would suggest the grep() function and some of its additional arguments that make it a pleasure to use.

grep("stringofinterest",names(dataframeofinterest),ignore.case=TRUE,value=TRUE)

without the argument value=TRUE you will only get a vector of index positions where the match occurred.

[share](https://stackoverflow.com/a/5672116)[improve this answer](https://stackoverflow.com/posts/5672116/edit)

Sort rows in a dataframe based on one column content

population[order(population$age),]

order(..., na.last = TRUE, decreasing = FALSE,

method = c("auto", "shell", "radix"))

sorted\_res<-outcome[order(outcome[,11],outcome[,2]),]

> str1 <- "How many words are in this sentence"

> sapply(strsplit(str1, " "), length)

[1] 7

my.data.frame <- subset(data , V1 > 2 | V2 < 4)

An alternative solution that mimics the behavior of this function and would be more appropriate for inclusion within a function body:

new.data <- data[ which( data$V1 > 2 | data$V2 < 4) , ]

if(length(grep(word(condition,1),names(outcome),ignore.case=TRUE))!=0& length(grep(word(condition,2),names(outcome),ignore.case=TRUE))!=0){…

**Programming assignment 3**

Part 1



Part 2

Best=function(state,condition){

best.R

**best<-function(state,condition){**

##Sorry that I took the liberty to change the name of the second function argument, for the sake of clarity

## Assuming that stringr library has been loaded because we will use the word() function

## CHECKS if State and condition are a match. If not, error message via stop function.

**acc\_cond<-c("Heart attack","Heart failure","Pneumonia")**

**pal<-sapply(strsplit(condition, " "), length)**

**if(length(grep(state,outcome[,7], ignore.case=TRUE))==0){**

**stop("Sorry, State is not a match")**

**} else {outcome\_s<-subset(outcome,State==state)**

**##Checks if the condition is pneumonia, heart attack or heart failure**

**if(pal==1){**

**if(length(grep(word(condition),names(outcome),ignore.case=TRUE))!=0){**

**colnum<-23**

**##because pneumonia is the condition**

**} else { stop("Sorry, condition is not a match")**

**}**

**} else if(pal>=3) { stop("Sorry, condition is not a match")**

**} else { if(length(grep(word(condition,1),names(outcome),ignore.case=TRUE))!=0& length(grep(word(condition,2),names(outcome),ignore.case=TRUE))!=0){**

**if(grep(word(condition,2),acc\_cond,ignore.case=TRUE)==1){**

**colnum<-11**

**} else { colnum<-17**

**}**

**}**

**else { stop("Sorry, condition is not a match")}**

**}**

**}**

**res<- subset(outcome\_s,select=c(2,7,colnum),na.rm=TRUE)**

**sorted\_res<-res[order(as.numeric(res[,3]),res[,1]),]**

**print(sorted\_res[1,1])**

**}**

**Assignment 3, part 3**

rankhospital.R

**rankhospital<-function(state,condition,num){**

**##Sorry that I took the liberty to change the name of the second function argument, for the sake of clarity**

**## Assuming that stringr library has been loaded because we will use the word() function**

**## CHECKS if State and condition are a match. If not, error message via stop function.**

**acc\_cond<-c("Heart attack","Heart failure","Pneumonia")**

**pal<-sapply(strsplit(condition, " "), length)**

**if(length(grep(state,outcome[,7], ignore.case=TRUE))==0){**

**stop("Sorry, State is not a match")**

**} else {outcome\_s<-subset(outcome,State==state)**

**##Checks if the condition is pneumonia, heart attack or heart failure**

**if(pal==1){**

**if(length(grep(word(condition),names(outcome),ignore.case=TRUE))!=0){**

**colnum<-23**

**##because pneumonia is the condition**

**} else { stop("Sorry, condition is not a match")**

**}**

**} else if(pal>=3) { stop("Sorry, condition is not a match")**

**} else { if(length(grep(word(condition,1),names(outcome),ignore.case=TRUE))!=0& length(grep(word(condition,2),names(outcome),ignore.case=TRUE))!=0){**

**if(grep(word(condition,2),acc\_cond,ignore.case=TRUE)==1){**

**colnum<-11**

**} else { colnum<-17**

**}**

**}**

**else { stop("Sorry, condition is not a match")}**

**}**

**}**

**res<- subset(outcome\_s,select=c(2,7,colnum),na.rm=TRUE)**

**write.csv(res,"res.csv")**

**clean\_res<-subset(res,res[,3]!="Not Available")**

**sorted\_res<-clean\_res[order(as.numeric(clean\_res[,3]),clean\_res[,1]),]**

**write.csv(sorted\_res,"sorted\_res.csv")**

**r\_s\_res<-sorted\_res**

**write.csv(r\_s\_res,"r\_s\_res.csv")**

**for(i in 1:nrow(r\_s\_res)){**

**r\_s\_res[i,4]<-i**

**colnames(r\_s\_res)[4]<-"Rank"**

**}**

**if(num=="best"){num<-as.numeric(1)**

**} else if (num=="worst") { num<-as.numeric(nrow(r\_s\_res)) } else {}**

**print(r\_s\_res[as.numeric(num),1])**

**}**

**rh("TX","heart failure",4)**

**Part 4 ranking hospitals in all states**

**Need to correct WORST!!**

**rankall <- function(condition, num = "best") {**

**##Sorry that I took the liberty to change the name of the first function argument, for the sake of clarity**

**## Assuming that stringr library has been loaded because we will use the word() function**

**## CHECKS if condition is a match. If not, error message via stop function.**

**pal<-sapply(strsplit(condition, " "), length)**

**##Checks if the condition is pneumonia, heart attack or heart failure**

**if(pal==1){**

**if(length(grep(word(condition),names(outcome),ignore.case=TRUE))!=0){**

**colnum<-23**

**##because pneumonia is the condition**

**} else { stop("Sorry, condition is not a match")**

**}**

**} else if(pal>=3) { stop("Sorry, condition is not a match")**

**} else { if(length(grep(word(condition,1),names(outcome),ignore.case=TRUE))!=0& length(grep(word(condition,2),names(outcome),ignore.case=TRUE))!=0){**

**if(grep(word(condition,2),acc\_cond,ignore.case=TRUE)==1){**

**colnum<-11**

**} else { colnum<-17**

**}**

**}**

**else { stop("Sorry, condition is not a match")}**

**}**

**x<-character()**

**y<-character()**

**for (i in 1:length(unique(outcome[,7]))) {**

**##subsetting main data frame by state**

**outc\_st<-subset(outcome,outcome[,7]==unique(outcome[,7])[i],na.rm=TRUE)**

**##Eliminating "Not Available" rows**

**cl\_outc\_st<-subset(outc\_st,outc\_st[,colnum]!="Not Available")**

**##In case num is best or worst**

**if(num=="best"){num1<-as.numeric(1)**

**} else if (num=="worst") { num1<-nrow(cl\_outc\_st)**

**} else {num1<-num}**

**##sorting subset by condition outcome**

**sor\_outc\_st<-cl\_outc\_st[order(as.numeric(cl\_outc\_st[,colnum]),cl\_outc\_st[,2]),]**

**##Selecting row in question**

**int\_res<-sor\_outc\_st[num1,]**

**write.csv(int\_res,"int\_res.csv")**

**x[i]<-int\_res[,2]**

**y[i]<- unique(outcome[,7])[i]**

**}**

**comp<-data.frame(x, y, stringsAsFactors=FALSE)**

**names(comp)<-c("Hospital","State")**

**write.csv(comp,"comp.csv")**

**comp\_s<-comp[order(comp$State),]**

**write.csv(comp\_s,"comp\_s.csv")**

**comp\_s**

**}**

r <- rankall("pneumonia", "worst")

as.character(subset(r, State == "NJ")$Hospital)

write.csv(sor\_outc\_st,paste("sor\_out\_st",i,".csv",sep=""))

print("Hola")

**Congratulations! You passed!**

Next Item

Question 1

Correct

1 / 1 points

**1. Question 1**

What result is returned by the following code?



1

best("SC", "heart attack")

Question 2

Correct

1 / 1 points

**2. Question 2**

What result is returned by the following code?



1

best("NY", "pneumonia")

Question 3

Correct

1 / 1 points

**3. Question 3**

What result is returned by the following code?



1

best("AK", "pneumonia")

Question 4

Correct

1 / 1 points

**4. Question 4**

What result is returned by the following code?



1

rankhospital("NC", "heart attack", "worst")

Question 5

Correct

1 / 1 points

**5. Question 5**

What result is returned by the following code?



1

rankhospital("WA", "heart attack", 7)

Question 6

Correct

1 / 1 points

**6. Question 6**

What result is returned by the following code?



1

rankhospital("TX", "pneumonia", 10)

Question 7

Correct

1 / 1 points

**7. Question 7**

What result is returned by the following code?



1

rankhospital("NY", "heart attack", 7)

Question 8

Correct

1 / 1 points

**8. Question 8**

What result is returned by the following code?



1

2

3

r <- rankall("heart attack", 4)

as.character(subset(r, state == "HI")$hospital)

Question 9

Correct

1 / 1 points

**9. Question 9**

What result is returned by the following code?



1

2

3

r <- rankall("pneumonia", "worst")

as.character(subset(r, state == "NJ")$hospital)

Question 10

Correct

1 / 1 points

**10. Question 10**

What result is returned by the following code?

For week commencing

cut(as.Date(fecha),"week",start.on.monday=FALSE)

Adding columns to a data table

DT[,w:=z^2]

DT[,m:={tmp<-(x+z);log2(tmp+5)}] ##Executes the last command, so adds another column m with the value of the last command ("it returns the evaluation of the last statement")

Install from swirl

install\_from\_swirl("Getting and Cleaning Data")

|  |  |
| --- | --- |
| Comparison {base} |  |

## Relational Operators

### Description

Binary operators which allow the comparison of values in atomic vectors.

### Usage

x < y

x > y

x <= y

x >= y

x == y

x != y

dplyr tidyr readr

The author of tidyr, Hadley Wickham, discusses his philosophy of tidy data in

| his 'Tidy Data' paper:

http://vita.had.co.nz/papers/tidy-data.pdf

# Don't change any of the code below. Just type submit()

# when you think you understand it. If you find it

# confusing, you're absolutely right!

result2 <-

arrange(

filter(

summarize(

group\_by(cran,

package),

count = n(),

unique = n\_distinct(ip\_id),

countries = n\_distinct(country),

avg\_bytes = mean(size)

),

countries > 60

),

desc(countries),

avg\_bytes

)

print(result2)

# Read the code below, but don't change anything. As

# you read it, you can pronounce the %>% operator as

# the word 'then'.

#

# Type submit() when you think you understand

# everything here.

result3 <-

cran %>%

group\_by(package) %>%

summarize(count = n(),

unique = n\_distinct(ip\_id),

countries = n\_distinct(country),

avg\_bytes = mean(size)

) %>%

filter(countries > 60) %>%

arrange(desc(countries), avg\_bytes)

# Print result to console

print(result3)

| To help drive the point home, let's work through a few more examples of chaining.

ID DateRange1Start DateRange1End Value1 DateRange2Start DateRange2End Value2 DateRange3Start DateRange3End Value3

1 1/1/90 3/1/90 4.4 4/5/91 6/7/91 6.2 5/5/95 6/6/96 3.3

reshape(dat, idvar="ID", direction="long",

varying=list(Start=c(2,5,8), End=c(3,6,9), Value=c(4,7,10)),

v.names = c("DateRangeStart", "DateRangeEnd", "Value") )

#-------------

ID time DateRangeStart DateRangeEnd Value

1.1 1 1 1/1/90 3/1/90 4.4

1.2 1 2 4/5/91 6/7/91 6.2

1.3 1 3 5/5/95 6/6/96 3.3

<http://garrettgman.github.io/tidying/>

**ext\_wh<-function(){**

**alsip<-read.csv("alsip ph count.csv")**

**alsip2a<-subset(alsip,select=c(1,2,3,4,5,6,7,8,16))**

**alsip2b<-subset(alsip,select=c(1,2,3,9,10,11,12,13,16))**

**alsip3a<-gather(alsip2a,"lot","lotnum",4:8)**

**write.csv(alsip3a,"alsip3a.csv")**

**alsip3b<-gather(alsip2b,"cs\_lot","cs",4:8)**

**write.csv(alsip3b,"alsip3b.csv")**

**##alsip4a<-subset(alsip3a,lotnum!="NA")**

**##alsip4b<-subset(alsip3b,cs!="NA")**

**alsip4a<-alsip3a**

**alsip4b<-alsip3b**

**write.csv(alsip4b,"alsip4b.csv")**

**write.csv(alsip4a,"alsip4a.csv")**

**alsip5a<-mutate(alsip4a,index=substr(alsip4a$lot,nchar(alsip4a$lot),nchar(alsip4a$lot)))**

**write.csv(alsip5a,"alsip5a.csv")**

**alsip5b<-mutate(alsip4b,index=substr(alsip4b$cs\_lot,nchar(alsip4b$cs\_lot),nchar(alsip4b$cs\_lot)))**

**write.csv(alsip5b,"alsip5b.csv")**

**alsip6<-merge(alsip5a,alsip5b,by=c("Load","index","X70"),all=FALSE)**

**write.csv(alsip6,"alsip6.csv")**

**## ~~alsip7<-group\_by(alsip6,tolower(Item),Load,lotnum)~~**

**alsip7<-group\_by(arrange(mutate(alsip6,ni=substr(Item.x,1,2)),ni,Load,lotnum),ni,Load,lotnum, tolower(Item.x),ignore.case=TRUE)**

**##(or toupper)**

**##solo al hacer la siguiente instrucción se ponen todos los items en mayúscula (o minúscula), curioso.**

**##Y el colname cambia a tolower(Item)**

**alsip8<-summarize(alsip7,tot=sum(cs))**

**write.csv(alsip8,"alsip8.csv")**

**alsip9<-mutate(alsip8,pounds=tot\*70)**

**alsip9bis<-alsip9[complete.cases(alsip9),]**

**alsip9\_sim<-select(alsip9bis,-ignore.case)**

**alsip9\_fin<-summarize(group\_by(alsip9\_sim,Load,lotnum),sum(tot),sum(pounds))**

**write.csv(alsip9bis,"alsip9bis.csv")**

**write.csv(alsip9\_sim,"alsip9\_sim.csv")**

**write.csv(alsip9\_fin,"alsip9\_fin.csv")**

**alsip9\_no\_load<-summarize(group\_by(alsip9bis,ni,lotnum),sum(tot),sum(pounds))**

**write.csv(alsip9\_no\_load,"alsip9\_no\_load.csv")**

**}**

mutate(alsip9\_no\_load,lb=

**Cleaning DATA week1  
Congratulations! You passed!**

Next Item

Question 1

Correct

1 / 1

point

## 1. Question 1

The American Community Survey distributes downloadable data about United States communities. Download the 2006 microdata survey about housing for the state of Idaho using download.file() from here:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Fss06hid.csv>

and load the data into R. The code book, describing the variable names is here:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FPUMSDataDict06.pdf>

How many properties are worth $1,000,000 or more?



164



47



31



53

**Correct**

Question 2

Correct

1 / 1

point

## 2. Question 2

Use the data you loaded from Question 1. Consider the variable FES in the code book. Which of the "tidy data" principles does this variable violate?



Numeric values in tidy data can not represent categories.



Each variable in a tidy data set has been transformed to be interpretable.



Tidy data has no missing values.



Tidy data has one variable per column.

**Correct**

Question 3

Correct

1 / 1

point

## 3. Question 3

Download the Excel spreadsheet on Natural Gas Aquisition Program here:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FDATA.gov_NGAP.xlsx>

Read rows 18-23 and columns 7-15 into R and assign the result to a variable called:



1

dat

What is the value of:



1

sum(dat$Zip\*dat$Ext,na.rm=T)

(original data source: <http://catalog.data.gov/dataset/natural-gas-acquisition-program>)



NA



0



154339



36534720

**Correct**

Question 4

Correct

1 / 1

point

## 4. Question 4

Read the XML data on Baltimore restaurants from here:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Frestaurants.xml>

How many restaurants have zipcode 21231?



127

**Correct**



17



156



100

Question 5

Incorrect

0 / 1

point

## 5. Question 5

The American Community Survey distributes downloadable data about United States communities. Download the 2006 microdata survey about housing for the state of Idaho using download.file() from here:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Fss06pid.csv>

using the fread() command load the data into an R object



1

DT

The following are ways to calculate the average value of the variable



1

pwgtp15

broken down by sex. Using the data.table package, which will deliver the fastest user time?



mean(DT[DT$SEX==1,]$pwgtp15); mean(DT[DT$SEX==2,]$pwgtp15)

**This should not be selected**



sapply(split(DT$pwgtp15,DT$SEX),mean)



rowMeans(DT)[DT$SEX==1]; rowMeans(DT)[DT$SEX==2]



mean(DT$pwgtp15,by=DT$SEX)



DT[,mean(pwgtp15),by=SEX]



tapply(DT$pwgtp15,DT$SEX,mean)

## Sorting by Multiple Columns (chartio.com)

**In some cases, it may be desired to sort by multiple columns. Thankfully, doing so is very simple with the previously described methods.**

**To sort multiple columns using vector names, simply add additional arguments to the order() function call as before:**

***# Sort by vector name [z] then [x]***

**dataframe[**

**with(dataframe, order(z, x)),**

**]**

**Similarly, to sort by multiple columns based on column index, add additional arguments to order() with differing indices:**

***# Sort by column index [1] then [3]***

**dataframe[**

**order( dataframe[,1], dataframe[,3] ),**

**]**

**crsst<-function() {**

**##To turn results table into cross table values**

**##Update source file name!!**

**ovw<-read.csv("OvW WeekC Oct 14.csv")**

**colnames(ovw)<-c("LocTyp","Wc","OvW")**

**##Converting to.Date and sorting by descending date**

**wc<-as.Date(ovw$Wc,"%d-%b-%y")**

**ovw\_d<-data.frame(loc\_typ=ovw$LocTyp,wc=wc,sobre=ovw$OvW)**

**ovw\_ord<-arrange(ovw\_d,desc(wc),loc\_typ)**

**povw<- data.frame(loc\_typ=ovw\_ord$loc\_typ,wc=wc,sobre=ovw\_ord$sobre)**

**write.csv(povw,"povw.csv")**

**loc<-unique(povw[,1])**

**weekc<-unique(povw[,2])**

**write.csv(weekc,"weekc.csv")**

**##Initializing the dataframe**

**crsstbl<-data.frame(matrix(ncol=length(weekc),nrow=length(loc)))**

**for (i in 1:length(loc)){**

**for (j in 1:length(weekc)){**

**if(length(subset(povw,loc\_typ==loc[i]&wc==weekc[j])[,3])!=0){**

**crsstbl[i,j]<-subset(povw,loc\_typ==loc[i]&wc==weekc[j])[,3]**

**} else{crsstbl[i,j]<-"NA"}**

**}**

**}**

**for (i in 1:length(loc)){ rownames(crsstbl)[i]<-as.character(loc[i]) }**

**for (j in 1:length(weekc)){colnames(crsstbl)[j]<-as.character(weekc[j])}**

**write.csv(crsstbl,"crsstbl.csv")**

**}**

[**Check whether value exist in one data frame or not**](https://stackoverflow.com/questions/13774773/check-whether-value-exist-in-one-data-frame-or-not)

I have two data frames (A,B)having same column names(C), but can have different unique values in that column. I want to check if 'value' in column (C) in data frame (A) exists in data frame (B).

A = data.frame(C=c(1,2,3,4))

B = data.frame(C=c(1,3,4,7))

In above example, I want to check if '2' is present in B or not Is there any one liner without loop, as I have pretty big files and would have to check this at every line.

setdiff may be of interest – [James](https://stackoverflow.com/users/269476/james) [Dec 8 '12 at 13:01](https://stackoverflow.com/questions/13774773/check-whether-value-exist-in-one-data-frame-or-not#comment18943627_13774773)

Use %in% as follows

A$C %in% B$C

Which will tell you which values of column C of A are in B.

What is returned is a logical vector. In the specific case of your example, you get:

A$C %in% B$C

# [1] TRUE FALSE TRUE TRUE

Which you can use as an index to the rows of A or as an index to A$C to get the actual values:

# as a row index

A[A$C %in% B$C, ] # note the comma to indicate we are indexing rows

# as an index to A$C

A$C[A$C %in% B$C]

[1] 1 3 4 # returns all values of A$C that are in B$C

We can negate it too:

A$C[!A$C %in% B$C]

[1] 2   # returns all values of A$C that are NOT in B$C

If you want to know if a specific value is in B$C, use the same function:

2 %in% B$C # "is the value 2 in B$C ?"

# FALSE

A$C[2] %in% B$C # "is the 2nd element of A$C in B$C ?"

# FALSE

getwd()

[1] "C:/Users/javier.lores/OneDrive - EVANS FOOD PRODUCTS/Documents/R"

setwd("..")

getwd()

[1] "C:/Users/javier.lores/OneDrive - EVANS FOOD PRODUCTS/Documents"

setwd(".")

getwd()

[1] "C:/Users/javier.lores/OneDrive - EVANS FOOD PRODUCTS/Documents"

setwd("./R")

[1] "C:/Users/javier.lores/OneDrive - EVANS FOOD PRODUCTS/Documents/R"

# Append two more function calls to accomplish the following:

#

# 1. Use group\_by() (from dplyr) to group the data by part and

# sex, in that order.

#

# 2. Use mutate to add two new columns, whose values will be

# automatically computed group-by-group:

#

# \* total = sum(count)

# \* prop = count / total

#

sat %>%

select(-contains("total")) %>%

gather(part\_sex, count, -score\_range) %>%

separate(part\_sex, c("part", "sex")) %>%

group\_by(part,sex) %>%

mutate(total=sum(count),

prop=count/total

) %>% print

FROM

sat

# A tibble: 6 x 10

score\_range read\_male read\_fem read\_total math\_male math\_fem math\_total

<chr> <int> <int> <int> <int> <int> <int>

1 700-800 40151 38898 79049 74461 46040 120501

2 600-690 121950 126084 248034 162564 133954 296518

3 500-590 227141 259553 486694 233141 257678 490819

4 400-490 242554 296793 539347 204670 288696 493366

5 300-390 113568 133473 247041 82468 131025 213493

6 200-290 30728 29154 59882 18788 26562 45350

# ... with 3 more variables: write\_male <int>, write\_fem <int>,

# write\_total <int>

A tibble: 36 x 6

# Groups: part, sex [6]

score\_range part sex count total prop

<chr> <chr> <chr> <int> <int> <dbl>

1 700-800 read male 40151 776092 0.0517

2 600-690 read male 121950 776092 0.157

3 500-590 read male 227141 776092 0.293

4 400-490 read male 242554 776092 0.313

5 300-390 read male 113568 776092 0.146

6 200-290 read male 30728 776092 0.0396

7 700-800 read fem 38898 883955 0.0440

8 600-690 read fem 126084 883955 0.143

9 500-590 read fem 259553 883955 0.294

10 400-490 read fem 296793 883955 0.336

# ... with 26 more rows

a <- c(rep("A", 3), rep("B", 3), rep("C",2))

b <- c(1,1,2,4,1,1,2,2)

df <-data.frame(a,b)

duplicated(df)

[1] FALSE TRUE FALSE FALSE FALSE TRUE FALSE TRUE

> df[duplicated(df), ]

a b

2 A 1

6 B 1

8 C 2

> df[!duplicated(df), ]

a b

1 A 1

3 A 2

4 B 4

5 B 1

7 C 2

"C:/Users/JAVIER~1.LOR/ONEDRI~1/DOCUME~1/R/R-35~1.1"

HDF5 interface to R

### Installation

To install this package, start R and enter:

## try http:// if https:// URLs are not supported

source("https://bioconductor.org/biocLite.R")

biocLite("rhdf5")

### Documentation

To view documentation for the version of this package installed in your system, start R and enter:

browseVignettes("rhdf5")

Github App token

secret, I suppose

2abd9c85850d5ba43640a18d851c3a8d03ac09f4

kosmos

(https://auth0.com/docs/connections/social/github)

**https://github.com/r-lib/httr/blob/master/demo/oauth2-github.r**

|  |
| --- |
| myapp <- oauth\_app("github", |
|  | key = "c5b8d1b6921f50e98414", |
|  | secret = "595248eae6590aee0b19bbf1a84a248cac023390") |

dir<-"https://d396qusza40orc.cloudfront.net/getdata%2Fwksst8110.for"

example<-read.fwf(dir,widths=c(10,9,4,9,4,9,4,9,4),skip=4)

> col4<-as.numeric(example[,4])

> sum(col4)

[1] 32426.7

**ongratulations! You passed!**

Next Item

Question 1

Correct

1 / 1

point

## 1. Question 1

Register an application with the Github API here <https://github.com/settings/applications>. Access the API to get information on your instructors repositories (hint: this is the url you want "https://api.github.com/users/jtleek/repos"). Use this data to find the time that the datasharing repo was created. What time was it created?

This tutorial may be useful (<https://github.com/hadley/httr/blob/master/demo/oauth2-github.r>). You may also need to run the code in the base R package and not R studio.



2014-03-05T16:11:46Z



2012-06-20T18:39:06Z



2013-11-07T13:25:07Z

**Correct**



2013-08-28T18:18:50Z

Question 2

Incorrect

0 / 1

point

## 2. Question 2

The sqldf package allows for execution of SQL commands on R data frames. We will use the sqldf package to practice the queries we might send with the dbSendQuery command in RMySQL.

Download the American Community Survey data and load it into an R object called



1

acs

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Fss06pid.csv>

Which of the following commands will select only the data for the probability weights pwgtp1 with ages less than 50?



sqldf("select pwgtp1 from acs")

**This should not be selected**



sqldf("select pwgtp1 from acs where AGEP \lt< 50")



sqldf("select \* from acs")



sqldf("select \* from acs where AGEP \lt< 50 and pwgtp1")

Question 3

Correct

1 / 1

point

## 3. Question 3

Using the same data frame you created in the previous problem, what is the equivalent function to unique(acs$AGEP)



sqldf("select unique AGEP from acs")



sqldf("select distinct AGEP from acs")

**Correct**



sqldf("select AGEP where unique from acs")



sqldf("select distinct pwgtp1 from acs")

Question 4

Correct

1 / 1

point

## 4. Question 4

How many characters are in the 10th, 20th, 30th and 100th lines of HTML from this page:

http://biostat.jhsph.edu/~jleek/contact.html

(Hint: the nchar() function in R may be helpful)



45 92 7 2



45 31 7 31



43 99 8 6



43 99 7 25



45 31 2 25



45 0 2 2



45 31 7 25

**Correct**

Question 5

Correct

1 / 1

point

## 5. Question 5

Read this data set into R and report the sum of the numbers in the fourth of the nine columns.

<https://d396qusza40orc.cloudfront.net/getdata%2Fwksst8110.for>

Original source of the data: <http://www.cpc.ncep.noaa.gov/data/indices/wksst8110.for>

(Hint this is a fixed width file format)



101.83



35824.9



222243.1



32426.7

**Correct**



28893.3



36.5

quantile(pack\_sum$unique,probs=0.99)

99%

465

filter(pack\_sum,unique>465)

by\_package <- group\_by(cran, package)

pack\_sum <- summarize(by\_package,

count = n(),

unique = n\_distinct(ip\_id),

countries = n\_distinct(country),

avg\_bytes = mean(size))

# Here's the new bit, but using the same approach we've

# been using this whole time.

top\_countries <- filter(pack\_sum, countries > 60)

result1 <- arrange(top\_countries, desc(countries), avg\_bytes)

# Print the results to the console.

print(result1)

Is equivalent to…

result2 <-

arrange(

filter(

summarize(

group\_by(cran,

package

),

count = n(),

unique = n\_distinct(ip\_id),

countries = n\_distinct(country),

avg\_bytes = mean(size)

),

countries > 60

),

desc(countries),

avg\_bytes

)

print(result2)

equivalent to

result3 <-

cran %>%

group\_by(package) %>%

summarize(count = n(),

unique = n\_distinct(ip\_id),

countries = n\_distinct(country),

avg\_bytes = mean(size)

) %>%

filter(countries > 60) %>%

arrange(desc(countries), avg\_bytes)

# Print result to console

print(result3)

library("dplyr")

#>

#> Attaching package: 'dplyr'

#> The following objects are masked from 'package:stats':

#>

#> filter, lag

#> The following objects are masked from 'package:base':

#>

#> intersect, setdiff, setequal, union

db <- src\_sqlite(tempfile(), create = TRUE)

iris2 <- copy\_to(db, iris)

**vec <- pull(iris2, Species)**

**head(vec)**

#> [1] "setosa" "setosa" "setosa" "setosa" "setosa" "setosa"

collect(select(iris2, Species))[[1]]

library(magrittr)

library(dplyr)

iris2 %>%

select(Species) %>%

extract2(1)

rlang::last\_error()

## Steve C Walker

# Remove (or replace) everything before or after a specified character in R strings

FEBRUARY 13, 2013

No time to explain this one, but here’s an example:

> x <- 'aabb.ccdd'

> sub('.\*', '', x)

[1] ""

> sub('bb.\*', '', x)

[1] "aa"

> sub('.\*bb', '', x)

[1] ".ccdd"

> sub('\\..\*', '', x)

[1] "aabb"

> sub('.\*\\.', '', x)

[1] "ccdd"

No need for [substring](http://help.r-enthusiasts.com/library/base/html/substr.html), just use [gsub](http://help.r-enthusiasts.com/library/base/html/grep.html):

gsub( " .\*$", "", dob )

# [1] "9/9/43" "9/17/88" "11/21/48"

A space (), then any character (.) any number of times (\*) until the end of the string ($). See [?regex](http://help.r-enthusiasts.com/library/base/html/regex.html) to learn regular expressions.

[share](https://stackoverflow.com/a/15895183)[improve this answer](https://stackoverflow.com/posts/15895183/edit)

answered Apr 9 '13 at 6:51

[Romain Francois](https://stackoverflow.com/users/499163/romain-francois)

**14.5k**23769

* 2

The only advice I can share here is that a sub is enough since there is only one string end position. – [Wiktor Stribiżew](https://stackoverflow.com/users/3832970/wiktor-stribi%c5%bcew) [Oct 12 '16 at 21:50](https://stackoverflow.com/questions/15895050/using-gsub-to-extract-character-string-before-white-space-in-r#comment67296027_15895183)

**> b<-substr(as.character(chi\_yp[7,33]),1,nchar(as.character(chi\_yp[7,33])))**

**> b**

**[1] "6,267.00"**

**> c<-as.numeric(sub(",.\*$","",b))\*1000+as.numeric(sub('.\*\\,', '', b))**

**> c**

**[1] 6267**

sat %>% select(-contains(total))

parse\_number(class\_5)

update(this\_moment, hours = 8, minutes = 34, seconds = 55)

vs

this\_moment<-update(this\_moment, hours = 8, minutes = 34, seconds = 55)

http://en.wikipedia.org/wiki/List\_of\_tz\_database\_time\_zones

pull fuciona con factors si es un data.frame, no si es un tibble!!

LIBRARIES

library(tidyr)

library(readr) ## parse\_number… incluso de factors

library(stringr)

library(varhandle)

library(assertthat)

library(dplyr)

library(lubridate)

##help(package = lubridate)

library(quantmod)

library(reshape2)

library(magrittr)

library(grDevices)

library(RColorBrewer)

library(ggplot2)

library(lattice)

library(datasets)

library(swirl)

package ‘dotCall64’ successfully unpacked and MD5 sums checked

package ‘spam’ successfully unpacked and MD5 sums checked

package ‘maps’ successfully unpacked and MD5 sums checked

package ‘fields’ successfully unpacked and MD5 sums checked

Package ‘fields’ loaded correctly!

Package ‘jpeg’ loaded correctly!

Package ‘datasets’ loaded correctly!

Quiz week 3 getting and cleaning data

setwd("C:/Users/javier.lores/OneDrive - EVANS FOOD PRODUCTS/Documents/R")

> acs\_id<-read.csv("acs\_id.csv")

> agricultureLogical<-as.logical(acs\_id$ACR==3&acs\_id$AGS==6)

> length(agricultureLogical)

[1] 6496

> which(agricultureLogical)

[1] **125 238 262** 470 555 568 608 643 787 808 824 849 952 955 1033

[16] 1265 1275 1315 1388 1607 1629 1651 1856 1919 2101 2194 2403 2443 2539 2580

[31] 2655 2680 2740 2838 2965 3131 3133 3163 3291 3370 3402 3585 3652 3852 3862

[46] 3912 4023 4045 4107 4113 4117 4185 4198 4310 4343 4354 4448 4453 4461 4718

[61] 4817 4835 4910 5140 5199 5236 5326 5417 5531 5574 5894 6033 6044 6089 6275

[76] 6376 6420

foto<-readJPEG("Fjeff.jpg",native=TRUE) ##after downloading the pic in my R folder

quantile(foto,probs=.30)

30%

-15259150

> quantile(foto,probs=.80)

80%

-10575416

edu<-read.csv("edu.csv")

gdp\_c<-read.csv("gdp.csv",skip=4,nrows=190)

todo<-merge(gdp\_c,edu,by.x="X",by.y="CountryCode",all.x=FALSE,no.dups=TRUE)

dim(todo)

[1] 189 40

todo\_sor<-arrange(todo,desc(X.1))

todo\_sor[13,4]

[1] St. Kitts and Nevis

190 Levels: Afghanistan Albania Algeria Angola Antigua and Barbuda ... Zimbabwe

todo\_g<-group\_by(todo,Income.Group)

View(todo\_g)

todo\_g\_sum<-summarize(todo\_g,mean=mean(X.1))

View(todo\_g\_sum)

todo\_q<-select(filter(todo,Income.Group=="Lower middle income",X.1<=38),X,X.1,Income.Group)

View(todo\_q)

**Congratulations! You passed!**

Question 1

Correct

1 / 1

point

## 1. Question 1

The American Community Survey distributes downloadable data about United States communities. Download the 2006 microdata survey about housing for the state of Idaho using download.file() from here:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Fss06hid.csv>

and load the data into R. The code book, describing the variable names is here:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FPUMSDataDict06.pdf>

Create a logical vector that identifies the households on greater than 10 acres who sold more than $10,000 worth of agriculture products. Assign that logical vector to the variable agricultureLogical. Apply the which() function like this to identify the rows of the data frame where the logical vector is TRUE.

which(agricultureLogical)

What are the first 3 values that result?



403, 756, 798



59, 460, 474



25, 36, 45



125, 238,262

**Correct**

Question 2

Correct

1 / 1

point

## 2. Question 2

Using the jpeg package read in the following picture of your instructor into R

<https://d396qusza40orc.cloudfront.net/getdata%2Fjeff.jpg>

Use the parameter native=TRUE. What are the 30th and 80th quantiles of the resulting data? (some Linux systems may produce an answer 638 different for the 30th quantile)



-16776430 -15390165



-15259150 -10575416

**Correct**



10904118 -594524



-10904118 -10575416

Question 3

Correct

1 / 1

point

## 3. Question 3

Load the Gross Domestic Product data for the 190 ranked countries in this data set:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FGDP.csv>

Load the educational data from this data set:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FEDSTATS_Country.csv>

Match the data based on the country shortcode. How many of the IDs match? Sort the data frame in descending order by GDP rank (so United States is last). What is the 13th country in the resulting data frame?

Original data sources:

<http://data.worldbank.org/data-catalog/GDP-ranking-table>

<http://data.worldbank.org/data-catalog/ed-stats>



189 matches, 13th country is St. Kitts and Nevis

**Correct**



190 matches, 13th country is St. Kitts and Nevis



189 matches, 13th country is Spain



190 matches, 13th country is Spain



234 matches, 13th country is Spain



234 matches, 13th country is St. Kitts and Nevis

Question 4

Correct

1 / 1

point

## 4. Question 4

What is the average GDP ranking for the "High income: OECD" and "High income: nonOECD" group?



23, 45



133.72973, 32.96667



32.96667, 91.91304

**Correct**



23, 30



30, 37



23.966667, 30.91304

Question 5

Correct

1 / 1

point

## 5. Question 5

Cut the GDP ranking into 5 separate quantile groups. Make a table versus Income.Group. How many countries

are Lower middle income but among the 38 nations with highest GDP?



0



12



5

**Correct**



13

flr\_date<-filter(flr,!grepl('tes',type,ignore.case=TRUE))

count(flr\_date,type)

View(count(flr\_date,type))

grep("stringofinterest",names(dataframeofinterest),ignore.case=TRUE,value=TRUE)

without the argument value=TRUE you will only get a vector of index positions where the match occurred.

With the stringr package, you can modify the pattern with one of the built in modifier functions (see `?modifiers). For example since we are matching a fixed string (no special regular expression characters) but want to ignore case, we can do

str\_detect(colnames(iris), fixed("species", ignore\_case=TRUE))

Or you can use the (?i) case insensitive [modifier](https://www.regular-expressions.info/modifiers.html)

str\_detect(colnames(iris), "(?i)species")

QUIZ: Cleaning data, course 3, week 4,

##Selecting only the observations that contain numbers in the "Gross…" column using the parse\_number function and filter out for NA

gdp<-read.csv("gdp.csv")

View(gdp)

class(gdp[,2])

[1] "factor"

class(gdp[8,2])

[1] "factor"

gdp\_s<-arrange(gdp,gdp[,2])

View(gdp\_s)

gdp\_r<-mutate(gdp,rank=parse\_number(Gross.domestic.product.2012))

Warning: 5 parsing failures.

row # A tibble: 5 x 4 col row col expected actual expected <int> <int> <chr> <chr> actual 1 3 NA a number Ranking row 2 237 NA a number . col 3 238 NA a number . expected 4 239 NA a number . actual 5 240 NA a number .

View(gdp\_r)

gdp\_rf<-filter(gdp\_r,is.na(rank)==FALSE)

View(gdp\_rf)

##To calculate the mean of the GDP after removing commas

mean(parse\_number(gdp\_rf$X.3))

[1] 377652.4

or

mean(as.numeric(as.character(gsub(",","",gdp\_rf$X.3))))

[1] 377652.4

##to count the number of countries whose name starts with United

grep("^United",gdp\_rf$X.2)

[1] 1 6 32

length(grep("^United",gdp\_rf$X.2))

[1] 3

##to merge data frames and count how many countrys' fiscal year ends in June

todo<-merge(edu,gdp\_rf,by.x="CountryCode",by.y="X",all.x=FALSE,no.dups=TRUE)

View(todo)

grep("Fiscal year end(.\*)[Jj]une",todo$Special.Notes)

[1] 9 16 29 51 65 89 96 133 140 152 159 175 189

length(grep("Fiscal year end(.\*)[Jj]une",todo$Special.Notes))

[1] 13

##more resrtrictive

length(grep("Fiscal year end: June",todo$Special.Notes))

[1] 13

##to know how many times AMZN stock value sampled in 2012 and how many on Mondays, after a few tests

dim(sampleTimes)

NULL

View(sampleTimes)

st<-sampleTimes

class(st)

[1] "Date"

st2012<-st[grep("2012",st)]

View(st2012)

length(st2012)

[1] 250

wday(st2012[1])

[1] 3

wday(st2012[1],label=TRUE)

[1] Tue

Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat

length(grep("Mon",wday(st2012,label=TRUE)))

[1] 47

**Congratulations! You passed!**

Next Item

Question 1

Correct

1 / 1

point

## 1. Question 1

The American Community Survey distributes downloadable data about United States communities. Download the 2006 microdata survey about housing for the state of Idaho using download.file() from here:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Fss06hid.csv>

and load the data into R. The code book, describing the variable names is here:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FPUMSDataDict06.pdf>

Apply strsplit() to split all the names of the data frame on the characters "wgtp". What is the value of the 123 element of the resulting list?



"wgtp" "15"



"wgtp"



"" "15"

**Correct**



"wgt" "15"

Question 2

Correct

1 / 1

point

## 2. Question 2

Load the Gross Domestic Product data for the 190 ranked countries in this data set:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FGDP.csv>

Remove the commas from the GDP numbers in millions of dollars and average them. What is the average?

Original data sources:

<http://data.worldbank.org/data-catalog/GDP-ranking-table>



387854.4



381615.4



377652.4

**Correct**



381668.9

Question 3

Correct

1 / 1

point

## 3. Question 3

In the data set from Question 2 what is a regular expression that would allow you to count the number of countries whose name begins with "United"? Assume that the variable with the country names in it is named countryNames. How many countries begin with United?



grep("\*United",countryNames), 2



grep("^United",countryNames), 3

**Correct**



grep("United$",countryNames), 3



grep("^United",countryNames), 4

Question 4

Correct

1 / 1

point

## 4. Question 4

Load the Gross Domestic Product data for the 190 ranked countries in this data set:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FGDP.csv>

Load the educational data from this data set:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FEDSTATS_Country.csv>

Match the data based on the country shortcode. Of the countries for which the end of the fiscal year is available, how many end in June?

Original data sources:

<http://data.worldbank.org/data-catalog/GDP-ranking-table>

<http://data.worldbank.org/data-catalog/ed-stats>



13

**Correct**



8



16



7

Question 5

Correct

1 / 1

point

## 5. Question 5

You can use the quantmod (<http://www.quantmod.com/>) package to get historical stock prices for publicly traded companies on the NASDAQ and NYSE. Use the following code to download data on Amazon's stock price and get the times the data was sampled.

library(quantmod)

amzn = getSymbols("AMZN",auto.assign=FALSE)

sampleTimes = index(amzn)

How many values were collected in 2012? How many values were collected on Mondays in 2012?



251,51



365, 52



252, 50



250, 47

**Correct**

read.table(dat, header = FALSE, sep = ",",

col.names = paste0("V",seq\_len(7)), fill = TRUE)

read.table**(**x\_train, header = FALSE, sep = "",

col.names = c**(**acc1,acc2,acc3,w1,w2,w3,paste0("V",seq\_len(561))**))**

y\_train<-scan("y\_train.txt",character(),quote="")

##Read 7352 items

##View(y\_train)

s\_train<-scan("subject\_train.txt",character(),quote="")

##Read 2947 items

xx\_train<-read.table("X\_train.txt", header = FALSE, sep = "", nrows=7352)

##dim(xx\_train)

##[1] 7352 561

##xx\_train[7352,561]

##[1] 0.03669484

##o bien (parece que da mas garantia; he hecho comprobaciones de elementos de ambas y son iguales, and it is safe because it is explained in the README.txt that one row is one set)

## xxbis\_train<-read.table("X\_train.txt", header = FALSE, sep = "", col.names=feat\_ps)

## dim(xxbis\_train)

> ?cbind

> yxx\_train<-cbind(y\_train,xx\_train)

> syxx\_train<-cbind(s\_train,yxx\_train)

> dim(syxx\_train)

[1] 7352 563

> str(xyxx\_train)

'data.frame': 7352 obs. of 563 variables:

$ s\_train: Factor w/ 21 levels "1","11","14",..: 1 1 1 1 1 1 1 1 1 1 ...

$ y\_train: Factor w/ 6 levels "1","2","3","4",..: 5 5 5 5 5 5 5 5 5 5 ...

$ V1 : num 0.289 0.278 0.28 0.279 0.277 ...

$ V2 : num -0.0203 -0.0164 -0.0195 -0.0262 -0.0166 ...

$ V3 : num -0.133 -0.124 -0.113 -0.123 -0.115 ...

$ V4 : num -0.995 -0.998 -0.995 -0.996 -0.998 ...

…

…

[list output truncated]

> syxx\_train[7352,563]

[1] 0.03669484

> y\_test<-scan("y\_test.txt",character(),quote="")

Read 2947 items

> s\_test<-scan("subject\_test.txt",character(),quote="")

Read 2947 items

> xx\_test<-read.table("X\_test.txt", header = FALSE, sep = "", nrows=2947)

> dim(xx\_test)

[1] 2947 561

> syxx\_test<-cbind(s\_test,y\_test,xx\_test)

> dim(syxx\_test)

[1] 2947 563

##kind of restarting

feat<-read.delim("features.txt",header=FALSE,sep="")

##> class(feat)

##[1] "data.frame"

##> dim(feat)

##[1] 561 3

##> View(feat)

##> feat\_ps<-substr(as.character(feat\_p$V1),3,nchar(as.character(feat\_p$V1))-1)

##> View(feat\_ps)

##> library(stringr)

##> feat\_ps<-str\_trim(substr(as.character(feat\_p$V1),3,nchar(as.character(feat\_p$V1))-1))

##> View(feat\_ps)

##> length(feat\_ps)

##[1] 561

> colnames(syxx\_test)<-c("subject","activity",feat[,2])

> colnames(syxx\_train)<-c("subject","activity",feat[,2])

##I am going to bind the columns of what the authors call the inertial signals, although it seems that it would not really be necessary because there are no mean or sd calculated variables in those files and therefore nothing to extract

sigt1<-read.table("./UCI HAR Dataset/train/Inertial Signals/body\_acc\_x\_train.txt", header = FALSE, sep = "", col.names= paste0("baxt",seq\_len(128)))

> sigt2<-read.table("./UCI HAR Dataset/train/Inertial Signals/body\_acc\_y\_train.txt", header = FALSE, sep = "", col.names= paste0("bayt",seq\_len(128)))

> sigt3<-read.table("./UCI HAR Dataset/train/Inertial Signals/body\_acc\_z\_train.txt", header = FALSE, sep = "", col.names= paste0("bazt",seq\_len(128)))

> sigt4<-read.table("./UCI HAR Dataset/train/Inertial Signals/body\_gyro\_x\_train.txt", header = FALSE, sep = "", col.names= paste0("bgxt",seq\_len(128)))

> sigt5<-read.table("./UCI HAR Dataset/train/Inertial Signals/body\_gyro\_y\_train.txt", header = FALSE, sep = "", col.names= paste0("bgyt",seq\_len(128)))

> sigt6<-read.table("./UCI HAR Dataset/train/Inertial Signals/body\_gyro\_z\_train.txt", header = FALSE, sep = "", col.names= paste0("bgzt",seq\_len(128)))

> sigt7<-read.table("./UCI HAR Dataset/train/Inertial Signals/total\_acc\_x\_train.txt", header = FALSE, sep = "", col.names= paste0("taxt",seq\_len(128)))

> sigt8<-read.table("./UCI HAR Dataset/train/Inertial Signals/total\_acc\_y\_train.txt", header = FALSE, sep = "", col.names= paste0("tayt",seq\_len(128)))

> sigt9<-read.table("./UCI HAR Dataset/train/Inertial Signals/total\_acc\_z\_train.txt", header = FALSE, sep = "", col.names= paste0("tazt",seq\_len(128)))

sigp1<-read.table("./UCI HAR Dataset/test/Inertial Signals/body\_acc\_x\_test.txt", header = FALSE, sep = "", col.names= paste0("baxt",seq\_len(128)))

sigp2<-read.table("./UCI HAR Dataset/test/Inertial Signals/body\_acc\_y\_test.txt", header = FALSE, sep = "", col.names= paste0("bayt",seq\_len(128)))

sigp3<-read.table("./UCI HAR Dataset/test/Inertial Signals/body\_acc\_z\_test.txt", header = FALSE, sep = "", col.names= paste0("bazt",seq\_len(128)))

sigp4<-read.table("./UCI HAR Dataset/test/Inertial Signals/body\_gyro\_x\_test.txt", header = FALSE, sep = "", col.names= paste0("bgxt",seq\_len(128)))

sigp5<-read.table("./UCI HAR Dataset/test/Inertial Signals/body\_gyro\_y\_test.txt", header = FALSE, sep = "", col.names= paste0("bgyt",seq\_len(128)))

sigp6<-read.table("./UCI HAR Dataset/test/Inertial Signals/body\_gyro\_z\_test.txt", header = FALSE, sep = "", col.names= paste0("bgzt",seq\_len(128)))

sigp7<-read.table("./UCI HAR Dataset/test/Inertial Signals/total\_acc\_x\_test.txt", header = FALSE, sep = "", col.names= paste0("taxt",seq\_len(128)))

sigp8<-read.table("./UCI HAR Dataset/test/Inertial Signals/total\_acc\_y\_test.txt", header = FALSE, sep = "", col.names= paste0("tayt",seq\_len(128)))

sigp9<-read.table("./UCI HAR Dataset/test/Inertial Signals/total\_acc\_z\_test.txt", header = FALSE, sep = "", col.names= paste0("tazt",seq\_len(128)))

##tot\_train<-cbind(syxx\_train, sigt1,sigt2,sigt3,sigt4,sigt5,sigt6,sigt7,sigt8,sigt9)

##tot\_test<-cbind(syxx\_test, sigp1,sigp2,sigp3,sigp4,sigp5,sigp6,sigp7,sigp8,sigp9)

##Adding one column with one character value, adding a constant, with the type of subject partition (train/test)

##tot\_test\_t<-cbind(tot\_test, paste0("test"))

##tot\_train\_t<-cbind(tot\_train, paste0("train"))

##binding both data sets

tot\_bin<-rbind(tot\_train\_t,tot\_test\_t)

tot\_bin1<-rbind(sigt1,sigp1)

tot\_bin2<-rbind(sigt2,sigp2)

tot\_bin3<-rbind(sigt3,sigp3)

tot\_bin4<-rbind(sigt4,sigp4)

tot\_bin5<-rbind(sigt5,sigp5)

tot\_bin6<-rbind(sigt6,sigp6)

tot\_bin7<-rbind(sigt7,sigp7)

tot\_bin8<-rbind(sigt8,sigp8)

tot\_bin9<-rbind(sigt9,sigp9)

tot\_bin\_all<-cbind(tot\_bin1,tot\_bin2,tot\_bin3,tot\_bin4,tot\_bin5,tot\_bin6,tot\_bin7,tot\_bin8,tot\_bin9)

tot\_syxx<-rbind(syxx\_train,syxx\_test)

dim(tot\_syxx)

[1] 10299 563

dim(tot\_bin\_all)

[1] 10299 1152

tot\_tot<-cbind(tot\_syxx,tot\_bin\_all)

dim(tot\_tot)

[1] 10299 1715

##showing the names of the activities rather than having just a number

act\_nam<-read.delim("activity\_labels.txt",header=FALSE,sep="")

colnames(act\_nam)<-c("activity","activityname")

data\_for\_analysis<-merge(tot\_tot,act\_nam,all.x=TRUE)

colnames(data\_for\_analysis)[1716]

[1] "activityname"

##I should have been merging all the time, not cbinding!!?? No, because there are no keys

data\_ms<-data\_for\_analysis[,grep("activ|subje|mean|std",colnames(data\_for\_analysis))]

##Eliminate columns data are not calculated means of a variable (meanFreq)

des<-c(grep("freq",colnames(data\_ms),ignore.case=TRUE))

data\_ms\_2<-data\_ms[,-des]

##remove dashes from col names

b<- mapply(gsub,"-","",colnames(data\_ms\_2))

data\_ms\_2\_col<-data\_ms\_2

colnames(data\_ms\_2\_col)<-b

##Just to simplify the name of the dataset for further manipulation

data\_f1<-data\_ms\_2\_col

##Convert values to numeric, because of the error messages I got when I tried to calculate means of columns

for(i in 1:nrow(data\_f1)){

for(j in 3:(ncol(data\_f1)-1)){

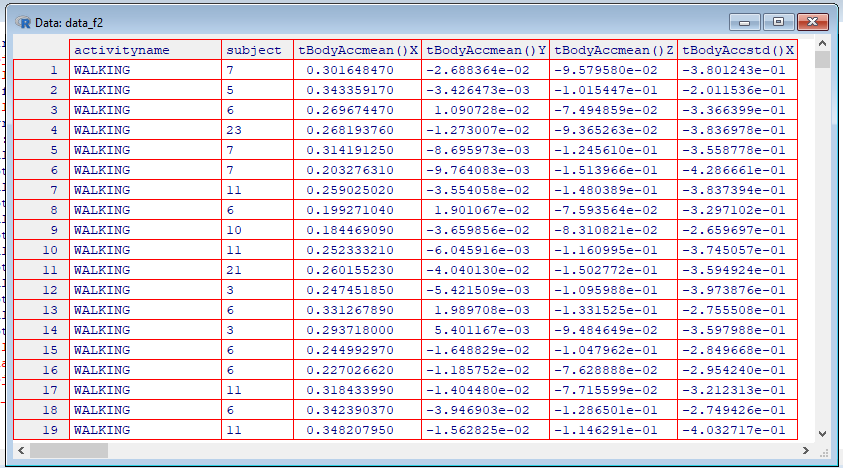
data\_f2[i,j]<-as.numeric(data\_f1[i,j])

}}

##Rename columns of data\_f2, back to the cleaned original variable names stored for the first time in b

nombres<-c("activityname","subject",colnames(data\_f1[,3:68]))

colnames(data\_f2)<-nombres



##That is the dataset requested, and I export it as a csv file

write.csv(data\_f2,"data\_f2.csv")

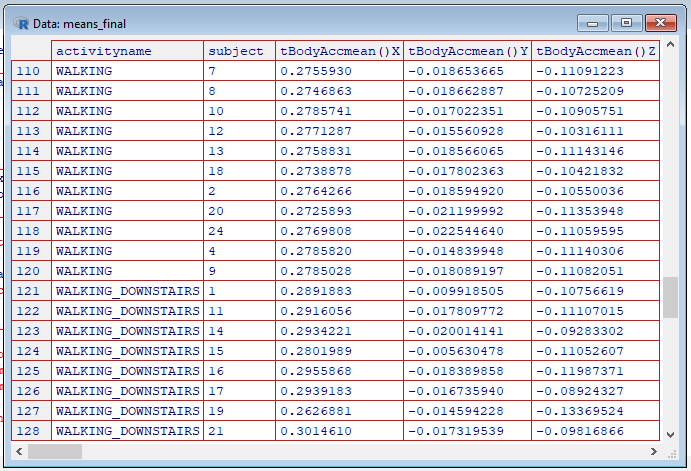
##Now the second part of the exercise…, Group

data\_f3<-group\_by(data\_f2,activityname,subject)

##Summarize all

means\_final<- summarise\_all(data\_f3,funs(mean))

View(means\_final)



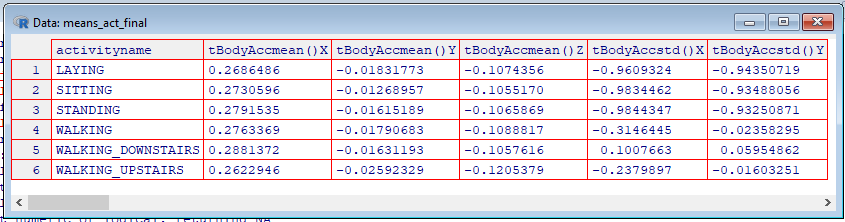
##That was the second dataset requested, that I export as csv file

write.csv(means\_final,"means\_final.csv")

##Means grouped by activity, all subjects collapsed

means\_act\_final<-summarize\_all(group\_by(select(data\_f2,-subject),activityname),funs(mean))

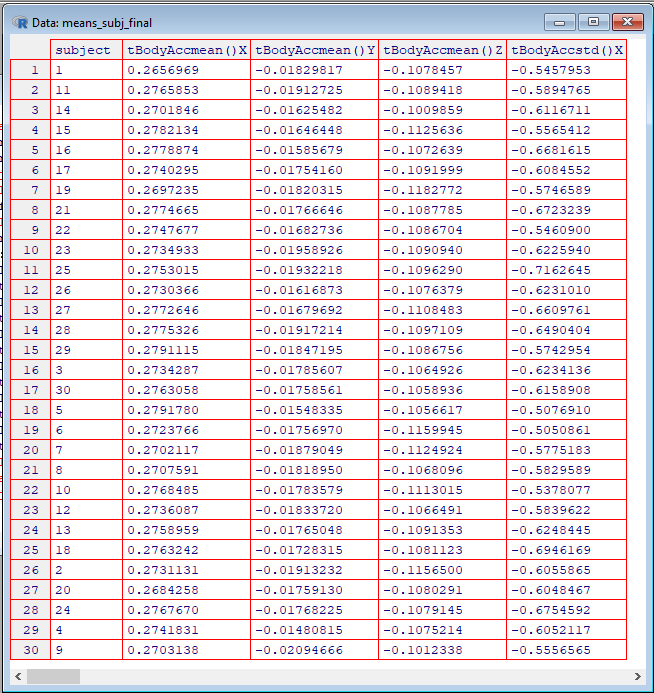
View(means\_act\_final)



##Means by subject, all activities collpased

means\_subj\_final<-summarize\_all(group\_by(select(data\_f2,-activityname),subject),funs(mean))

View(means\_subj\_final)



**https://github.com/thecmos/Coursera\_tidying**

## ERROR can't bind data because arguments have the same name: data\_ms<-select(data\_for\_analysis,activity,activityname,subject,as.character(grep("mean|std",as.character(colnames(data\_for\_analysis)),value=TRUE)))

To replace or remove part of a string in all elements of a vector

* **Using sapply**
* data$abstract <- sapply(data$abstract,
* function(x){gsub(pattern = "no abstract available",
* replacement = " ", x)})
* **Using mapply**
* data$abstract <- mapply(gsub, pattern = "no abstract available", replacement = " ", data$abstract)
* **Using the stringr package**
* library(stringr)

data$abstract <- str\_replace(data$abstract, "no abstract available",

" ")

Ideas not to type all different tot\_bins…

The direct solution to your question would be to use get with paste

for(i in 1:10) {

Object = get(paste0("Season", i))

Object[1] = 0

assign(paste0("Season", i), Object)

}

**But don't do this**.

It's a horrible use of R. As suggested in the comments, store results in lists:

Seasons = lapply(rep(10,10), rnorm) #Generate data

Seasons

Then apply functions:

Seasons = lapply(Seasons, replace, list=1, values=0)

**Here's a solution based on @joran's comment**

> set.seed(1) # for reproducibility

> # the following does the same as your `for` loop and returned value is a list

> Season.list <- replicate(10, rnorm(10, 0, 1), simplify=FALSE)

> # giving some names

> names(Season.list) <- paste0("Season", 1:length(Season.list))

> # setting first element to 1

> Season.list <- lapply(Season.list, function(x) {x[1] <- 0; x})

> list2env(Season.list, envir = .GlobalEnv) # will give you each `Season` as you want :D

**Also, another way to do it,**

> set.seed(1)

> Season <- replicate(10, rnorm(10, 0, 1)) # the returned object is a matrix

> colnames(Season) <- paste0("Season", 1:ncol(Season))

> Season[1,] <- 0

If you want to have a vector for each Season then use attach (not a good idea)

> attach(as.data.frame(Season))

> Season1

[1] 0.0000000 0.1836433 -0.8356286 1.5952808 0.3295078 -0.8204684 0.4874291 0.7383247 0.5757814 -0.3053884

> Season2

[1] 0.00000000 0.38984324 -0.62124058 -2.21469989 1.12493092 -0.04493361 -0.01619026 0.94383621 0.82122120

[10] 0.59390132

Or, if you want to keep the last duplicated column, you can do

temp <- temp[, !duplicated(colnames(temp), fromLast = TRUE)]

**To group CHI pellet types properly and graph financial impact of yield**

pellets<-function()**{**

##chi\_yp\_prev<-read.csv("TEST Chicago Daily YP.csv",skip=1)

chi\_yp\_prev<-read.csv("TEST Chicago Daily YP.csv")

**##Average pellet yields for 2017**

##Only 2017 data

chi\_yp17<-filter(chi\_yp\_prev,grepl("2017", chi\_yp\_prev$Yield))

m<-0

n<-0

for(i in 1:length(pel\_typ))**{**

for(j in 1:length(pel\_cols))**{**

for(k in 1:nrow(chi\_yp17))**{**

##Comprobar que hay coincidencia

if(length(grep(pel\_typ[i],as.character((chi\_yp17[k,pel\_cols[j]])),ignore.case=TRUE))!=0) **{**

##guardar el numero de la fila y columna en la que hay coincidencia

##en posiciones sucesivas de respectivos vectores

m<-m+1

colu[m]<-pel\_cols[j]

fila[m]<-k

typ[m]<-pel\_typ[i]

**}**

**}**

**}**

**}**

##build dataframe with every row in fila[m] and column colu[m] plus the next three

for(i in 1:m)**{**

#para cada fila seleccionada…

#sacar fecha

flr[i,1]<-as.Date(chi\_yp17[fila[i],2], "%m/%d/%Y")

d<-substr(as.character(chi\_yp17[fila[i],colu[i]]),1,nchar(as.character(chi\_yp17[fila[i],colu[i]])))

flr[i,2]<-d

for(j in 1:3)**{**

#para cada fila seleccionada…

#sacar columna del ítem

#sacar las tres columnas siguientes (las libras)

c<-parse\_number( chi\_yp17[fila[i],colu[i]+j] )

flr[i,j+2]<-c

##flr[i,j+2]<- chi\_yp17[fila[i],colu[i]+j]

**}**

**}**

colnames(flr)<-c("date","type","lbcut","lbcooked","lbobtained")

##write.csv(flr,"flr.csv")

flr\_tes<-filter(flr,!grepl('tes',type,ignore.case=TRUE))

flr\_tesdat<-filter(flr\_tes,grepl('201',date))

write.csv(flr\_tesdat,"flr\_tesdat.csv")

##remove NA to clean the original table completely

flr\_tesdat1<-select(flr\_tesdat,-lbcut)

flr\_tdc<-flr\_tesdat1[complete.cases(flr\_tesdat1),]

##step to help build yield\_key file

for\_key17<- summarize(group\_by(arrange(flr\_tdc,type),type),sumc=sum(lbcooked,na.rm=TRUE),sumo=sum(lbobtained,na.rm=TRUE),y=(sumo/sumc))

write.csv(for\_key17,"for\_key17.csv")

##Prepare for merging by selecting only the columns that we are interested in

flr\_key<-select(read.csv("yield\_key.csv"),type,fet)

##Merge, no need to write column names "…"

##The key is written manually on the yield\_key.csv file (at least the first time)

flr\_mrg<-merge(flr\_tdc,flr\_key,all.x=TRUE)

flr\_mrgg<-arrange(group\_by(flr\_mrg,fet,date),fet,date)

##write.csv(flr\_mrgg,"flr\_mrgg.csv")

alfa<-select(flr\_mrgg,-type)

beta<-mutate(alfa,weekc=as.character(ymd(date)+(1-wday(ymd(date)))))

write.csv(beta,"beta.csv")

gamma<-read.csv("beta.csv")[,3:6]

delta<-group\_by(gamma,fet,weekc)

flr\_mrgg\_weekc<-summarize(delta, sumc=sum(lbcooked,na.rm=TRUE),sumo=sum(lbobtained,na.rm=TRUE),yield=(sumo/sumc))

##write.csv(flr\_mrgg\_weekc,"flr\_mrgg\_weekc.csv")

##yield debe ser construido a partir de flr\_mrgg para que muestre las categorias refinadas

yield17<-arrange(summarize(group\_by(arrange(flr\_mrgg,fet),fet),sumc=sum(lbcooked,na.rm=TRUE),sumo=sum(lbobtained,na.rm=TRUE),y=(sumo/sumc)),desc(sumo))

write.csv(yield17,"yield17.csv")

##si quisiera sacar el top 6 en volumen

##yield\_fil17<-yield17[1:6,]

yield\_fil17<-yield17

##write.csv(yield\_fil17,"yield\_fil17.csv")

flr\_weekc\_fil17<-merge(flr\_mrgg\_weekc,yield\_fil17,by="fet")

##write.csv(flr\_weekc\_fil17,"flr\_weekc\_fil27.csv")

##compare list of pellets in for\_key and flr\_key (yield\_key.csv), and alert if there are "un-fetenized" pellet types

alert<-for\_key17$type%in%flr\_key$type

ifelse("FALSE"%in%(for\_key17$type%in%flr\_key$type),print(paste("Unmatched pellet type in for\_key17!",for\_key17$type[grep("FALSE",alert)])),print("Ok"))

**##Calculations on pellet yields for 2018**

##Only 2018 data

chi\_yp<-filter(chi\_yp\_prev,grepl("2018", chi\_yp\_prev$Yield))

##il\_yp<-tbl\_df(chi\_yp)

pel\_typ<-c("S-2","S2","S-3","S3","M-6","M6","M-7","M7","M-8","M8","M-5","ME","L6A","M5B","S3B","Str","CRA")

pel\_cols<-c(22,27,32,37,42,47,52,57,62)

fila<-vector()

typ<-vector()

colu<-vector()

flr<-data.frame()

m<-0

n<-0

for(i in 1:length(pel\_typ))**{**

for(j in 1:length(pel\_cols))**{**

for(k in 1:nrow(chi\_yp))**{**

##Comprobar que hay coincidencia

if(length(grep(pel\_typ[i],as.character((chi\_yp[k,pel\_cols[j]])),ignore.case=TRUE))!=0) **{**

##guardar el numero de la fila y columna en la que hay coincidencia

##en posiciones sucesivas de respectivos vectores

m<-m+1

colu[m]<-pel\_cols[j]

fila[m]<-k

typ[m]<-pel\_typ[i]

**}**

**}**

**}**

**}**

write.csv(colu,"colu.csv")

write.csv(fila,"fila.csv")

##build dataframe with every row in fila[m] and column colu[m] plus the next three

for(i in 1:m)**{**

#para cada fila seleccionada…

#sacar fecha

flr[i,1]<-as.Date(chi\_yp[fila[i],2], "%m/%d/%Y")

d<-substr(as.character(chi\_yp[fila[i],colu[i]]),1,nchar(as.character(chi\_yp[fila[i],colu[i]])))

flr[i,2]<-d

for(j in 1:3)**{**

#para cada fila seleccionada…

#sacar columna del ítem

#sacar las tres columnas siguientes (las libras)

c<-parse\_number( chi\_yp[fila[i],colu[i]+j] )

flr[i,j+2]<-c

##flr[i,j+2]<- chi\_yp[fila[i],colu[i]+j]

**}**

**}**

colnames(flr)<-c("date","type","lbcut","lbcooked","lbobtained")

write.csv(flr,"flr.csv")

flr\_tes<-filter(flr,!grepl('tes',type,ignore.case=TRUE))

flr\_tesdat<-filter(flr\_tes,grepl('201',date))

write.csv(flr\_tesdat,"flr\_tesdat.csv")

##remove NA to clean the original table completely

flr\_tesdat1<-select(flr\_tesdat,-lbcut)

flr\_tdc<-flr\_tesdat1[complete.cases(flr\_tesdat1),]

##step to help build yield\_key file

for\_key<- summarize(group\_by(arrange(flr\_tdc,type),type),sumc=sum(lbcooked,na.rm=TRUE),sumo=sum(lbobtained,na.rm=TRUE),y=(sumo/sumc))

write.csv(for\_key,"for\_key.csv")

##Prepare for merging by selecting only the columns that we are interested in

flr\_key<-select(read.csv("yield\_key.csv"),type,fet)

##Merge, no need to write column names "…"

##The key is written manually on the yield\_key.csv file (at least the first time)

flr\_mrg<-merge(flr\_tdc,flr\_key,all.x=TRUE)

flr\_mrgg<-arrange(group\_by(flr\_mrg,fet,date),fet,date)

write.csv(flr\_mrgg,"flr\_mrgg.csv")

alfa<-select(flr\_mrgg,-type)

beta<-mutate(alfa,weekc=as.character(ymd(date)+(1-wday(ymd(date)))))

write.csv(beta,"beta.csv")

gamma<-read.csv("beta.csv")[,3:6]

delta<-group\_by(gamma,fet,weekc)

flr\_mrgg\_weekc<-summarize(delta, sumc=sum(lbcooked,na.rm=TRUE),sumo=sum(lbobtained,na.rm=TRUE),yield=(sumo/sumc))

write.csv(flr\_mrgg\_weekc,"flr\_mrgg\_weekc.csv")

##yield debe ser construido a partir de flr\_mrgg para que muestre las categorias refinadas

yield<-arrange(summarize(group\_by(arrange(flr\_mrgg,fet),fet),sumc=sum(lbcooked,na.rm=TRUE),sumo=sum(lbobtained,na.rm=TRUE),y=(sumo/sumc)),desc(sumo))

write.csv(yield,"yield.csv")

##si quisiera sacar el top 6 en volumen

##yield\_fil<-yield[1:6,]

yield\_fil<-yield

##write.csv(yield\_fil,"yield\_fil.csv")

flr\_weekc\_fil<-merge(flr\_mrgg\_weekc,yield\_fil,by="fet")

write.csv(flr\_weekc\_fil,"flr\_weekc\_fil.csv")

##compare list of pellets in for\_key and flr\_key (yield\_key.csv), and alert if there are "un-fetenized" pellet types

##alert<-merge(for\_key,flr\_key,by.x="type",by.y="type",all.x=TRUE,sort=TRUE,no.dups=TRUE,incomparables=TRUE)

alert<-for\_key$type%in%flr\_key$type

ifelse("FALSE"%in%(for\_key$type%in%flr\_key$type),print(paste("Unmatched pellet type!",for\_key$type[grep("FALSE",alert)])),print("Ok"))

**}**

a<-substr(as.character(chi\_yp[7,23]),1,nchar(as.character(chi\_yp[7,23])))

> a

[1] "43,807"

> class(a)

[1] "character"

as.numeric(substr(a,1,1))

[1] 4

as.numeric(substr(a,1,1))\*3

[1] 12

> b<-sub('\\..\*', '', a)

> b

[1] "43,807"

> gsub(",.\*$","",a)

[1] "43"

**> b<-substr(as.character(chi\_yp[7,33]),1,nchar(as.character(chi\_yp[7,33])))**

**> b**

**[1] "6,267.00"**

**> c<-as.numeric(sub(",.\*$","",b))\*1000+as.numeric(sub('.\*\\,', '', b))**

**> c**

**[1] 6267**

b<-0

for(i 1:nchar(a)){

b<-b+10^(nchar(a)-i) \* ( as.numeric(substr(a,1,)))

chi\_yp\_pels<-filter(chi\_yp,pel\_typ%in%pel\_cols)

https://stats.stackexchange.com/questions/16796/reading-only-two-out-of-three-columns-with-read-csv

read.csv(file = "result1", sep = " ")[ ,c('col1', 'col2')]

impordata <- read.csv("skiplines.csv")

impordata <- subset(impordata,year>=1985)

subsetting and selecting all in one

ovw\_yn\_ps<-subset(ovw\_yn,subset=grepl("PS",ovw\_yn$loctype),select=!grepl("CR",colnames(ovw\_yn)))

ovw\_yn\_cr<-subset(ovw\_yn,subset=grepl("CR",ovw\_yn$loctype),select=!grepl("PS",colnames(ovw\_yn)))

yield\_factors\_R

dplyr

tidyr

readr

factors<-function(){

##reading tables and obtaining the tidy data set

OvW\_table<-read.csv("z) OvW Plant Wk.csv")

##View(OvW\_table)

y\_by\_wk<-read.csv("Copy Of 3 2) Y by week.csv")

##View(y\_by\_wk)

col\_ovw<-gsub("\\.","",colnames(OvW\_table))

col\_ovw<-gsub("^SumOfSumof","",col\_ovw)

colnames(OvW\_table)<-col\_ovw

col\_y<-gsub("\\.","",colnames(y\_by\_wk))

##View(col\_y)

colnames(y\_by\_wk)<-col\_y

colnames(y\_by\_wk)<-gsub("\_","",colnames(y\_by\_wk))

ovw\_y<-merge(OvW\_table,y\_by\_wk,by.x=c("WeekC","Location"),by.y=c("WeekC","Location"))

write.csv(ovw\_y,"ovw\_y.csv")

##ovw\_y1<-sapply(select(ovw\_y,-1,-2,-3,-7),parse\_number) ##does not do what I want

ovw\_y0<-select(ovw\_y,-1,-2,-3,-7)

ovw\_y1<-data.frame()

for(i in 1:nrow(ovw\_y0)){

for(j in 1:ncol(ovw\_y0)){

ovw\_y1[i,j]<-parse\_number(ovw\_y0[i,j])

}}

names(ovw\_y1)<-names(ovw\_y0)

ovw\_y2<-as.data.frame(cbind("date"=(as.character(ovw\_y[,1])), "location"=as.character(ovw\_y[,2]), "loctype"=as.character(ovw\_y[,3]),"locnav"= as.character(ovw\_y[,7]),ovw\_y1))

write.csv(ovw\_y2,"ovw\_y2.csv")

## colnames(ovw\_y2)

##[1] "date" "location" "loctype" "locnav"

##[5] "Overwpounds" "Totalpounds" "OvW" "PSWkY"

##[9] "CRWkY" "TOTWkY" "PSFriedLbsTot" "PSLbstot"

##[13] "PSWSeasLbsTot" "CRFriedLbsTot" "CRLbsTot" "AllFGLbsTot"

##[17] "Cheeselbs" "PSWLb" "CRWLb"

##Add the new calculated variables

ovw\_yn<-mutate(ovw\_y2, PS\_FG\_OvW= (PSLbstot)\*(1+(OvW)/100), PS\_Oil\_to\_Y\_PS=(PSLbstot)-(PSFriedLbsTot), PS\_OvW\_Oil\_to\_Y= (PS\_FG\_OvW)- (PSFriedLbsTot)+ (PSWLb), CR\_FG\_OvW= (CRLbsTot)\*(1+(OvW)/100), CR\_Oil\_to\_Y\_CR=(CRLbsTot)-(CRFriedLbsTot), CR\_OvW\_Oil\_to\_Y= (CR\_FG\_OvW)-(CRFriedLbsTot)+ (CRWLb), OilConFctPS= (PSWkY)-1+(Overwpounds)/(PSFriedLbsTot)+(PSWLb)/(PSFriedLbsTot), OvWFctPS=-(Overwpounds)/(PSFriedLbsTot), WFctPS=-(PSWLb)/(PSFriedLbsTot),OilConFctCR= (CRWkY)-1+(Overwpounds)/(CRFriedLbsTot)+(CRWLb)/(CRFriedLbsTot), OvWFctCR=-(Overwpounds)/(CRFriedLbsTot), WFctCR=-(CRWLb)/(CRFriedLbsTot))

write.csv(ovw\_yn,"ovw\_yn.csv")

##split in two tables, PS and CR

ovw\_yn\_ps<-subset(ovw\_yn,subset=grepl("PS",ovw\_yn$loctype),select=!grepl("CR",colnames(ovw\_yn)))

ovw\_yn\_cr<-subset(ovw\_yn,subset=grepl("CR",ovw\_yn$loctype),select=!grepl("PS",colnames(ovw\_yn)))

##a<-subset(ovw\_yn\_ps,as.character(ovw\_yn\_ps$date)=="02-Sep-18")

##convert to date and sort

aa<-arrange(mutate(ovw\_yn\_ps,date1=dmy(date)),desc(date1))

aa$date<-ymd(as.Date(aa$date,"%d-%b-%y"))

bb<-arrange(mutate(ovw\_yn\_cr,date1=dmy(date)),desc(date1))

bb$date<-ymd(as.Date(bb$date,"%d-%b-%y"))

write.csv(aa,"aa.csv")

write.csv(bb,"bb.csv")

##plot charts

plot(filter(aa,location=="CAL")$date,filter(aa,location=="CAL")$OilConFctPS,type="l",lwd=2,col="pink",main="Oil absorption as contribution to PS yield",ylab="",xlab="Date",ylim=c(0,max(aa$OilConFctPS)))

points(filter(aa,location=="TX")$date,filter(aa,location=="TX")$OilConFctPS,type="l",col="green")

points(filter(aa,location=="OH")$date,filter(aa,location=="OH")$OilConFctPS,type="l",col="blue")

points(filter(aa,location=="CHI")$date,filter(aa,location=="CHI")$OilConFctPS,type="l", lwd=2,col="orange")

points(filter(aa,location=="COI")$date,filter(aa,location=="COI")$OilConFctPS,type="l", lwd=1,col="red")

points(filter(aa,location=="MEX")$date,filter(aa,location=="MEX")$OilConFctPS,type="l",col="purple")

legend("bottom",lty=1:1,col=c("pink","green","blue","orange","red","purple"),legend=c("CAL","TX","OH","CHI","COI","MEX"),ncol=3,lwd=2)

abline(h=0.1,col="lightgray",lwd=1,lty="dotted")

abline(h=0.2,col=" lightgray ", lwd=1,lty="dotted")

abline(h=0.3,col=" lightgray ", lwd=1,lty="dotted")

abline(h=0.4,col=" lightgray ", lwd=1,lty="dotted")

abline(h=0.5,col=" lightgray ", lwd=1,lty="dotted")

plot(filter(bb,location=="CAL")$date,filter(bb,location=="CAL")$OilConFctCR,type="l",lwd=2,col="pink", main="Oil absorption as contribution to CR yield",ylab="",xlab="Date",ylim=c(min(bb$OilConFctCR),max(bb$OilConFctCR)))

points(filter(bb,location=="TX")$date,filter(bb,location=="TX")$OilConFctCR,type="l",col="green")

points(filter(bb,location=="OH")$date,filter(bb,location=="OH")$OilConFctCR,type="l",col="blue")

points(filter(bb,location=="CHI")$date,filter(bb,location=="CHI")$OilConFctCR,type="l", lwd=2,col="orange")

points(filter(bb,location=="COI")$date,filter(bb,location=="COI")$OilConFctCR,type="l", lwd=1,col="red")

points(filter(bb,location=="MEX")$date,filter(bb,location=="MEX")$OilConFctCR,type="l",col="purple")

legend("bottomleft",lty=1:1,col=c("pink","green","blue","orange","red","purple"),legend=c("CAL","TX","OH","CHI","COI","MEX"),ncol=3,lwd=2)

abline(h=-0.05,col="lightgray",lwd=1,lty="dotted")

abline(h=-0.1,col=" lightgray ", lwd=1,lty="dotted")

abline(h=-0.15,col=" lightgray ", lwd=1,lty="dotted")

abline(h=-0.2,col=" lightgray ", lwd=1,lty="dotted")

abline(h=-0.25,col=" lightgray ", lwd=1,lty="dotted")

##grid (nx=0,ny=NULL, lty = 1, col = "cornsilk2")

}

##ovw\_yn1<-ovw\_yn[,ovw\_yn$loctype %in% row.names(ovw\_yn)]

##d1[row.names(d1) %in% row.names(d2),]

Example(points)

Which of the following cliches captures the essence of graphics?

1: A rose by any other name smells as sweet

2: The apple doesn't fall far from the tree

3: A picture is worth a 1000 words

4: To err is human, to forgive divine

R command with, which evaluates "an R expression in an environment constructed from data".

In fact, this is an example of Simpson's paradox, or the

| Yule–Simpson effect. Wikipedia

| (http://en.wikipedia.org/wiki/Simpson%27s\_paradox) tells us that this "is a

| paradox in probability and statistics, in which a trend that appears in

| different groups of data disappears when these groups are combined."

Congrats! You've concluded exploring this lesson on principles of graphing.

| We hope you found it principally principled.

| Although ggplot2 bears a superficial similarity to lattice, it's generally

| easier and more intuitive to use. Its default mode makes many choices for you

| but you can still customize a lot. The package is based on a "grammar of

| graphics" (hence the gg in the name), so you can control the aesthetics of

| your plots. For instance, you can plot **conditioning graphs and panel plots** as

| we did in the lattice example.

| Congrats! You've concluded this plotting lesson. We hope you didn't find it

| plodding.

| Now we'll go through some quick examples of basic plotting before we delve

| into gory details. We'll use the dataset airquality (part of the library

| datasets) which we've loaded for you. This shows ozone and other air

| measurements for New York City for 5 months in 1973.

Variety is the spice of life.

You are quite good my friend!

All that practice is paying off!

Perseverance, that's the answer.

| The par() function is used to specify global graphics parameters that affect all plots in an R session. (Use dev.off or

| plot.new to reset to the defaults.) These parameters can be overridden when specified as arguments to specific plotting

| functions. These include las (the orientation of the axis labels on the plot), bg (background color), mar (margin size),

| oma (outer margin size), mfrow and mfcol (number of plots per row, column).

So you can add text, title, points, and lines to an existing plot. To add lines, you give a vector of x values and a

| corresponding vector of y values (or a 2-column matrix); the function lines just connects the dots. The function text adds

| text labels to a plot using specified x, y coordinates.

The function title adds annotations. These include x- and y- axis labels, title, subtitle, and outer margin. Two other

| annotating functions are mtext which adds arbitrary text to either the outer or inner margins of the plot and axis which

| adds axis ticks and labels. Another useful function is legend which explains to the reader what the symbols your plot uses

| mean.

type set equal to "n". This tells R to set up the plot but not to put the data in it.

> plot(airquality$Wind,airquality$Ozone,type="n")

Now use the R command points to plot May's wind and ozone (in that order) as solid blue triangles. You have to set the

| color and plot character with two separate arguments. Note we use points because we're adding to an existing plot.

You're close...I can feel it! Try it again. Or, type info() for more options.

with(may,points(Wind,Ozone,pch=17,col="blue")) ==

points(may$Wind,may$Ozone,col="blue",pch=17)

use the R command legend to clarify the plot and explain what it means. The function has a lot of arguments, but

| we'll only use 4. The first will be the string "topright" to tell R where to put the legend. The remaining 3 arguments will

| each be 2-long vectors created by R's concatenate function, e.g., c(). These arguments are pch, col, and legend. The first

| is the vector (17,8), the second ("blue","red"), and the third ("May","Other Months").

legend("topright",pch=c(17,8),col=c("blue","red"),legend=c("May","Other Months"))

abline(v=median(airquality$Wind),lty=2,lwd=2)

Use par with the parameter mfrow set equal to the vector (1,2) to set up the plot window for two plots side by side. You

| won't see a result.

> par(mfrow=c(1,2))

> plot(airquality$Wind,airquality$Ozone,main="Ozone and Wind")

> plot(airquality$Ozone,airquality$Solar.R,main="Ozone and Solar Radiation")

par(mfrow=c(1,3),mar=c(4,4,2,1),oma=c(0,0,2,0))

| Nice work!

| Margins are specified as 4-long vectors of integers. Each number tells how many lines of text to leave at each side. The

| numbers are assigned clockwise starting at the bottom. The default for the inner margin is c(5.1, 4.1, 4.1, 2.1) so you can

| see we reduced each of these so we'll have room for some outer text.

> plot(airquality$Wind,airquality$Ozone,main="Ozone and Wind")

> plot(airquality$Solar.R,airquality$Ozone,main="Ozone and Solar Radiation")

> plot(airquality$Temp,airquality$Ozone,main="Ozone and Temperature")



Not quite right, but keep trying. Or, type info() for more options.

mtext("Ozone and Weather in New York City", outer = TRUE).

Congrats! You've weathered this lesson nicely and passed out of the No!zone.

**Exploratory data analysis week 1 quiz - Congratulations! You passed!**

Question 1

*Correct*

1 / 1

point

## 1. Question 1

Which of the following is a principle of analytic graphics?



Only do what your tools allow you to do



Make judicious use of color in your scatterplots



Don't plot more than two variables at at time



Show box plots (univariate summaries)



Show multivariate data

Correct

Question 2

*Correct*

1 / 1

point

## 2. Question 2

What is the role of exploratory graphs in data analysis?



Axes, legends, and other details are clean and exactly detailed.



They are made for formal presentations.



Only a few are constructed.



They are typically made very quickly.

Correct

Question 3

*Correct*

1 / 1

point

## 3. Question 3

Which of the following is true about the base plotting system?



The system is most useful for conditioning plots



Plots are created and annotated with separate functions

Correct

Functions like 'plot' or 'hist' typically create the plot on the graphics device and functions like 'lines', 'text', or 'points' will annotate or add data to the plot.



Plots are typically created with a single function call



Margins and spacings are adjusted automatically depending on the type of plot and the data

Question 4

*Correct*

1 / 1

point

## 4. Question 4

Which of the following is an example of a valid graphics device in R?



A socket connection



The keyboard



A PDF file

Correct



A file folder

Question 5

*Correct*

1 / 1

point

## 5. Question 5

Which of the following is an example of a vector graphics device in R?



PNG



Postscript

Correct



JPEG



GIF



TIFF

Question 6

*Incorrect*

0 / 1

point

## 6. Question 6

Bitmapped file formats can be most useful for



Plots that require animation or interactivity

This should not be selected

In R, the SVG device is most suitable for including animations or interactivity



Plots that are not scaled to a specific resolution



Plots that may need to be resized



Scatterplots with many many points

Question 7

*Correct*

1 / 1

point

## 7. Question 7

Which of the following functions is typically used to add elements to a plot in the base graphics system?



boxplot()



plot()



points()

Correct



hist()

Question 8

*Correct*

1 / 1

point

## 8. Question 8

Which function opens the screen graphics device for the Mac?



quartz()

Correct



bitmap()



pdf()



png()

Question 9

*Correct*

1 / 1

point

## 9. Question 9

What does the 'pch' option to par() control?



the plotting symbol/character in the base graphics system

Correct



the size of the plotting symbol in a scatterplot



the orientation of the axis labels on the plot



the line width in the base graphics system

Question 10

*Correct*

1 / 1

point

## 10. Question 10

If I want to save a plot to a PDF file, which of the following is a correct way of doing that?



Construct the plot on the PNG device with png(), then copy it to a PDF with dev.copy2pdf().



Construct the plot on the screen device and then copy it to a PDF file with dev.copy2pdf()

Correct



Open the screen device with quartz(), construct the plot, and then close the device with dev.off().



Open the PostScript device with postscript(), construct the plot, then close the device

***A mi ni me funciona esto…***

> Sys.setenv(LANG = "fr")

> 2 + x

Erreur : objet 'x' introuvable

> Sys.setenv(LANG = "en")

> 2 + x

Error: object 'x' not found

FROM SWIRL…

boxplot(Ozone~Month,airquality,xlab="Month",ylab="Ozone (ppb)",col.axis="blue",col.lab="red")

title(main="Wind and Ozone in New York City")

with(airquality,plot(Wind,Ozone))

par()$pin ##same as par("pin")

[1] 5.760000 5.149583

par("pch") ## gives integer value of the default

[1] 1

?par

colors() function gives you a vector of colors by name.

The par() function is used to specify global graphics parameters that affect all plots in an R session. (Use dev.off or

plot.new to reset to the defaults.) These parameters can be overridden when specified as arguments to specific plotting

functions. These include las (the orientation of the axis labels on the plot), bg (background color), mar (margin size),

oma (outer margin size), mfrow and mfcol (number of plots per row, column).

So you can add text, title, points, and lines to an existing plot. To add lines, you give a vector of x values and a

corresponding vector of y values (or a 2-column matrix); the function lines just connects the dots. The function text adds

text labels to a plot using specified x, y coordinates.

The function title adds annotations. These include x- and y- axis labels, title, subtitle, and outer margin. Two other

annotating functions are mtext which adds arbitrary text to either the outer or inner margins of the plot and axis which

adds axis ticks and labels. Another useful function is legend which explains to the reader what the symbols your plot uses mean.

plot(airquality$Wind,airquality$Ozone,type="n")

title(main="Wind and Ozone in NYC")

may<-subset(airquality,Month==5)

Now use the R command points to plot May's wind and ozone (in that order) as solid blue triangles. You have to set the

color and plot character with two separate arguments. Note we use points because we're adding to an existing plot.

points(may$Wind,may$Ozone,col="blue",pch=17)

notmay<-subset(airquality,Month!=5)

Now use the R command points to plot these notmay's wind and ozone (in that order) as red snowflakes.

points(notmay$Wind,notmay$Ozone,col="red",pch=8)

Now we'll use the R command legend to clarify the plot and explain what it means. The function has a lot of arguments, but we'll only use 4. The first will be the string "topright" to tell R where to put the legend. The remaining 3 arguments will each be 2-long vectors created by R's concatenate function, e.g., c(). These arguments are pch, col, and legend. The first is the vector (17,8), the second ("blue","red"), and the third ("May","Other Months"). Try it now.

legend("topright",pch=c(17,8),col=c("blue","red"),legend=c("May","Other Months"))

Now add a vertical line at the median of airquality$Wind. Make it dashed (lty=2) with a width of 2.

abline(v=median(airquality$Wind),lty=2,lwd=2)

Use par with the parameter mfrow set equal to the vector (1,2) to set up the plot window for two plots side by side. You won't see a result.

par(mfrow=c(1,2))

plot(airquality$Wind, airquality$Ozone, main = "Ozone and Wind")

You nailed it! Good job!

Now for the second plot

Plot airquality$Ozone and airquality$Solar.R and use main to specify the title "Ozone and Solar Radiation".

plot(airquality$Ozone, airquality$Solar.R, main = "Ozone and Solar Radiation")

This one with 3 plots, to illustrate inner and outer margins. First, set up the plot window by typing par(mfrow = c(1, 3), mar = c(4, 4, 2, 1), oma = c(0, 0, 2, 0))

par(mfrow = c(1, 3), mar = c(4, 4, 2, 1), oma = c(0, 0, 2, 0))

Margins are specified as 4-long vectors of integers. Each number tells how many lines of text to leave at each side. The

numbers are assigned clockwise starting at the bottom. The default for the inner margin is c(5.1, 4.1, 4.1, 2.1) so you can see we reduced each of these so we'll have room for some outer text.

… … …

Since this is the main title, we specify it with the R command mtext. Call mtext with the string "Ozone and Weather in New York City" and the argument outer set equal to TRUE.

mtext("Ozone and Weather in New York City", outer = TRUE)

Keep up the great work!

Voila! Beautiful, right?

fit<-ln(y~x)

abline(fit,lwd=3,col="blue")

##pfeb<-##subset(read.table("power.txt",header=TRUE,sep=";"),as.character(Date)=="1/2/2007"|as.character(D##ate)=="2/2/2007")

##dim(pfeb)

##[1] 2880 9

##libraries that I will use

library(lubridate)

library(readr)

library(dplyr)

library(tidyr)

##read only data of the first two days of February

pfeb<-subset(read.table("household\_power\_consumption.txt",header=TRUE,sep=";"),as.character(Date)=="1/2/2007"|as.character(Date)=="2/2/2007")

##> dim(pfeb)

##[1] 2880 9

pfeb\_d<-mutate(pfeb,date=as.Date(dmy(as.character(Date))))

##pfeb\_d[3,10]

##[1] "2007-02-01"

##> dim(pfeb\_d)

##[1] 2880 10

##plot the histogram as required and copy in png file

hist(as.numeric(as.character(pfeb\_d$Global\_active\_power)),col="red",main="Global Active Power",xlab="Global Active Power (kilowatts)")

|  |
| --- |
| dev.copy(png, file = "plot1.png", bg = "white",height=480,width=480) |
|  |

dev.off()

##a<-paste(pfeb\_d[1,1],pfeb\_d[1,2])

##> strptime(a,format="%d/%m/%Y %H:%M:%S")

##[1] "2007-02-01 CST"

##> class(strptime(a,format="%d/%m/%Y %H:%M:%S"))

##strptime(pow[1,1],format="%d/%m/%Y %H:%M:%S")

##[1] "2007-02-01 CST"

##> as.Date(pow[1,1],format="%d/%m/%Y %H:%M:%S")

##[1] "2007-02-01"

##For the second graph better to convert all numeric variables to numeric class and store them into the new data frame “pow”, and merge date and time in one variable

pow<-data.frame()

for(i in 1:nrow(pfeb\_d)){

for(j in 1:1){

pow[i,j]<-paste(as.character(pfeb\_d[i,j]),as.character(pfeb\_d[i,j+1]))

}

for(j in 3:(ncol(pfeb\_d)-1)){

pow[i,j-1]<-parse\_number(pfeb\_d[i,j])

}}

names(pow)<-c("date\_time",names(pfeb\_d[3:9]))

powd<-mutate(pow,day=substr(wday(as.Date(date\_time, format="%d/%m/%Y %H:%M:%S"),label=TRUE),1,2))

##I realize that I have to add the time column back

powd[,10]<-pfeb[,2]

##names(powd[,10])<-"time\_fac" this doesn't work!!

names(powd)<-c(names(powd)[1:9],"time\_fac") ##this works

##create new column (mins): convert the time to a numeric variable with the number of minutes (hoursx60+min)

powdt<-mutate(powd,mins=as.numeric(substr(as.character(time\_fac),1,2))\*60+as.numeric(substr(as.character(time\_fac),4,5)))

##> dim(powdt)

##[1] 2880

##for(i in 1:nrow(powd)){

##powd[i,11]<- ##as.numeric(substr(as.character(powd[i,10]),1,2))\*60+as.numeric(substr(as.character(pfeb\_d[i,10]),4,5##))

##}

##as.numeric(substr(as.character(pfeb\_d[3,2]),1,2))\*60+as.numeric(substr(as.character(pfeb\_d[3,2]),4,5))

##plot step by step, first on screen, then copy to png

plot(1:nrow(powdt),powdt$Global\_active\_power,type="l",ylab="Global Active Power (kilowatts)",xlab="",**xaxt="n")**

axis(1,at=c(0,length(grep("^1/2",powdt$date\_time))+1,nrow(powdt)+1),labels=c("Thu","Fri","Sat"))

##axis(1,at=c(0,nrow(powdt)/2+1,nrow(powd)+1),labels=c("Thu","Fri","Sat"))

##axis(1,at=c(0,1441,2881),labels=c("Thu","Fri","Sat"))

|  |
| --- |
| dev.copy(png, file = "plot2.png", bg = "white",height=480,width=480) |
|  |

dev.off()

##plot(1:nrow(powdt),powdt$Global\_active\_power,type="l",ylab="Global Active Power (kilowatts)",xlab="",**xaxp=c(0,2881,2)**)

##plot(powdt$mins,powdt$Global\_active\_power,type="n")

##points(powdt$mins,powdt$Global\_active\_power,pch=".:")

##plot first on screen, then copy to png

plot(1:nrow(powdt),powdt$Sub\_metering\_1,type="n",ylab="Energy sub metering",xlab="",xaxt="n")

points(1:nrow(powdt),powdt$Sub\_metering\_1,type="l")

points(1:nrow(powdt),powdt$Sub\_metering\_2,type="l",col="red")

points(1:nrow(powdt),powdt$Sub\_metering\_3,type="l",col="blue")

axis(1,at=c(0,1441,2881),labels=c("Thu","Fri","Sat"))

legend("topright",lty=1:1,col=c("black","red","blue"),legend=c("Sub\_metering\_1","Sub\_metering\_2","Sub\_metering\_3"))

|  |
| --- |
| dev.copy(png, file = "plot3.png", bg = "white",height=480,width=480) |
|  |

dev.off()

##plot first on screen, then copy to png, all the plots created before with minor changes, like avoiding borderline on legend box

##Prepare the plot device for four graphs, tow rows and two columns

par(mfrow=c(2,2))

##Plot the four graphs, one by one, by rows

plot(1:nrow(powdt),powdt$Global\_active\_power,type="l",ylab="Global Active Power (kilowatts)",xlab="",xaxt="n")

axis(1,at=c(0,1441,2881),labels=c("Thu","Fri","Sat"))

plot(1:nrow(powdt),powdt$Voltage,type="l",ylab="Voltage",xlab="datetime",**xaxt="n")**

axis(1,at=c(0,1441,2881),labels=c("Thu","Fri","Sat"))

plot(1:nrow(powdt),powdt$Sub\_metering\_1,type="n",ylab="Energy sub metering",xlab="",xaxt="n")

points(1:nrow(powdt),powdt$Sub\_metering\_1,type="l")

points(1:nrow(powdt),powdt$Sub\_metering\_2,type="l",col="red")

points(1:nrow(powdt),powdt$Sub\_metering\_3,type="l",col="blue")

axis(1,at=c(0,1441,2881),labels=c("Thu","Fri","Sat"))

legend("topright",lty=1:1,col=c("black","red","blue"),bty="n",legend=c("Sub\_metering\_1","Sub\_metering\_2","Sub\_metering\_3"))

plot(1:nrow(powdt),powdt$Global\_reactive\_power,type="l",ylab="Global\_reactive\_power",xlab="datetime",**xaxt="n")**

axis(1,at=c(0,1441,2881),labels=c("Thu","Fri","Sat"))

|  |
| --- |
| dev.copy(png, file = "plot4.png", bg = "white",height=480,width=480) |
|  |

dev.off()

Other students:

https://github.com/GipFunny/ExData\_Plotting1

Very very compact

DateTime <- strptime(paste(pfeb$Date, pfeb$Time), "%d/%m/%Y %H:%M:%S")

Con esto salen los días de la semana…

<https://github.com/rts1988/ExData_Plotting1>

positions and writes tickers in a very smart way

<https://github.com/acuquel/ExData_Plotting1>

uses with

<https://github.com/KoenVermeulen91/PGA-EDA-Week1>

very well organized, nice and clean, why did I forget filter

         gsub("\\.", " ", data) Escape with a double backslash

         gsub("[.]", " ", data) Escape using character class

FROM SWIRL LATTICE

xyplot(Ozone~Wind|as.factor(Month),airquality,layout=c(5,1))

| That's correct! 27%

| Note that the default color and plotting character are back. What did the as.factor(Month) do?

1: Huh?

2: Randomly divided the data into 5 panels

3: Displayed the data by individual months

4: Displayed and labeled each subplot with the month's integer

v1

[1] -2.185287 1.101780 -2.716851 1.569850

| You're the best! 52%

| The first two numbers are the range of the x values of Group 1 and the last two numbers are the range of y values of Group

| 1. See how they match the values of the left panel (Group 1) in the plot. Now look at v2 which holds the comparable numbers

| for Group 2.

> v2

[1] -1.6066772 2.2205197 -0.1605085 2.0341048

| You're the best! 54%

| **Again, the values match the plot. That's reassuring.** We've copied some code from the slides for you. To see it, type myedit("plot1.R"). This will open your editor and display the R code in it.

p <- xyplot(y ~ x | f, panel = function(x, y, ...) {

panel.xyplot(x, y, ...) ## First call the default panel function for 'xyplot'

panel.abline(h = median(y), lty = 2) ## Add a horizontal line at the median

})

print(p)

invisible()

| The panel function has 3 arguments, x, y and ... . **This last stands for all other arguments (such as graphical parameters) you might want to include**. There are 2 lines in the panel function. Each invokes a panel method, the first to plot the data in each panel (panel.xyplot), the second to draw a horizontal line in each panel (panel.abline). Note the similarity ofthis last call to that of the base plotting function of the same name. 60%

| We've defined a function for you, pathtofile, which takes a filename as its argument. This makes sure R can find the file

| on your computer. Now run the R command source with two arguments. The first is the call to pathtofile with the string

| "plot1.R" as its argument and the second is the argument local set equal to TRUE. This command will run the code contained in plot1.R within the swirl environment so you can see what it does.

p2 <- xyplot(y ~ x | f, panel = function(x, y, ...) {

panel.xyplot(x, y, ...) ## First call default panel function

panel.lmline(x, y, col = 2) ## Overlay a simple linear regression line

})

print(p2)

invisible()

> str(diamonds)

Classes ‘tbl\_df’, ‘tbl’ and 'data.frame': 53940 obs. of 10 variables:

$ carat : num 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...

$ cut : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...

$ color : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...

$ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...

$ depth : num 61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...

$ table : num 55 61 65 58 58 57 57 55 61 61 ...

$ price : int 326 326 327 334 335 336 336 337 337 338 ...

$ x : num 3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...

$ y : num 3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...

$ z : num 2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...

| 72% So the data frame contains 10 pieces of information for each of 53940 diamonds. Run the R command table with diamonds$color as an argument.

> table(diamonds$color)

D E F G H I J

6775 9797 9542 11292 8304 5422 2808

table(diamonds$color,diamonds$cut)

Fair Good Very Good Premium Ideal

D 163 662 1513 1603 2834

E 224 933 2400 2337 3903

F 312 909 2164 2331 3826

G 314 871 2299 2924 4884

H 303 702 1824 2360 3115

I 175 522 1204 1428 2093

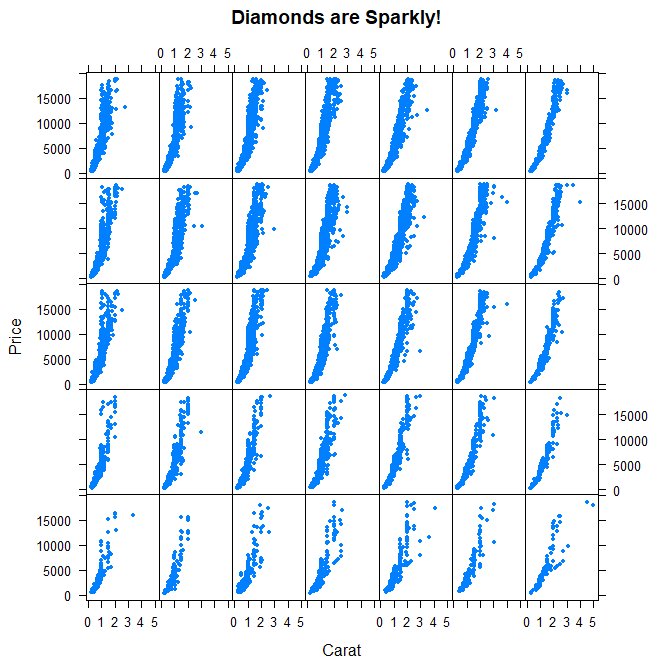
J 119 307 678 808 896

myxlab <- "Carat"

myylab <- "Price"

mymain <- "Diamonds are Sparkly!"

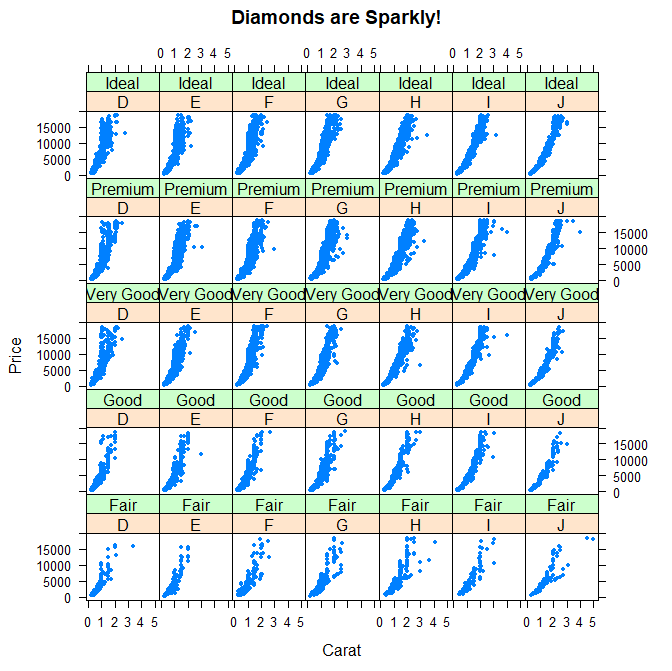
xyplot(price~carat|color\*cut,data=diamonds,**strip=FALSE**,pch=20,xlab=myxlab,ylab=myylab,main=mymain)



| Are colors defining the rows or columns of the plot?

1: rows 2: columns Selection: 2 | You're the best!

xyplot(price~carat|color\*cut,data=diamonds,pch=20,xlab=myxlab,ylab=myylab,main=mymain)



Congrats! We hope this lesson didn't leave you climbing the trellis.

From week 2 quiz

library(lattice)

library(datasets)

data(airquality)

p <- xyplot(Ozone ~ Wind | factor(Month), data = airquality)

Of course, color choice is secondary to your data and how you analyze it, but effectively using colors can enhance your

| plots and presentations, emphasizing the important points you're trying to convey.

To show you some options, here's a display of two color palettes that come with the grDevices package available to you. The

| left shows you some colors from the function heat.colors. Here low values are represented in red and as the values increase

| the colors move through yellow towards white. This is consistent with the physical properties of fire. The right display is

| from the function topo.colors which uses topographical colors ranging from blue (low values) towards brown (higher values).

sample(colors()) , sample(colors(),10)

colorRamp and colorRampPalette,

pal<-colorRamp(c("red","blue"))

| We don't see any output, but R has created the function pal which we can call with a single argument between 0 and 1. Call

| pal now with the argument 0.

> pal(0)

[,1] [,2] [,3]

[1,] 255 0 0

| We see a 1 by 3 array with 255 as the first entry and 0 in the other 2. This 3 long vector corresponds to red, green, blue

| (RGB) color encoding commonly used in televisions and monitors. In R, 24 bits are used to represent colors. Think of these

| 24 bits as 3 sets of 8 bits, each of which represents an intensity for one of the colors red, green, and blue.

The 255 returned from the pal(0) call corresponds to the largest possible number represented with 8 bits, so the vector

| (255,0,0) contains only red (no green or blue), and moreover, it's the highest possible value of red.

pal(1)

[,1] [,2] [,3]

[1,] 0 0 255

The function pal can take more than one argument. It returns one 3-long (or 4-long, but more about this later) vector for

| each argument. To see this in action, call pal with the argument seq(0,1,len=6).

> pal(seq(0,1,len=6))

[,1] [,2] [,3]

[1,] 255 0 0

[2,] 204 0 51

[3,] 153 0 102

[4,] 102 0 153

[5,] 51 0 204

[6,] 0 0 255

| 29%

| Six vectors (each of length 3) are returned. The i-th vector is identical to output that would be returned by the call

| pal(i/5) for i=0,...5. We see that the i-th row (for i=1,...6) differs from the (i-1)-st row in the following way. Its red

| entry is 51 = 255/5 points lower and its blue entry is 51 points higher.

So pal creates colors using the palette we specified when we called colorRamp. In this example none of pal's outputs will

| ever contain green since it wasn't in our initial palette.

The argument you pass to the returned function specifies the number of colors you want returned. Each element of the

| returned vector is a 24 bit number, represented as 6 hexadecimal characters, which range from 0 to F. This set of 6 hex

| characters represents the intensities of red, green, and blue, 2 characters for each color.

| To see this better, assign to the variable p1 the output of a call to colorRampPalette with the single argument,

| c("red","blue"). We'll compare it to our experiments using colorRamp.

> p1<-colorRampPalette(c("red","blue"))

| Now call p1 with the argument 2.

> p1(2)

[1] "#FF0000" "#0000FF"

| We see a 2-long vector is returned. The first entry FF0000 represents red. The FF is hexadecimal for 255, the same value

| returned by our call pal(0). The second entry 0000FF represents blue, also with intensity 255.

| Now call p1 with the argument 6. Let's see if we get the same result as we did when we called pal with the argument

| seq(0,1,len=6).

> p1(6)

[1] "#FF0000" "#CC0033" "#990066" "#650099" "#3200CC" "#0000FF"

| Now we get the 6-long vector (FF0000, CC0033, 990066, 650099, 3200CC, 0000FF). We see the two ends (FF0000 and 0000FF) are

| consistent with the colors red and blue. How about CC0033? Type 0xcc or 0xCC at the command line to see the decimal

| equivalent of this hex number. You must include the 0 before the x to specify that you're entering a hexadecimal number.

> 0xcc

[1] 204

| So 0xCC equals 204 and we can easily convert hex 33 to decimal, as in 0x33=3\*16+3=51. These were exactly the numbers we got

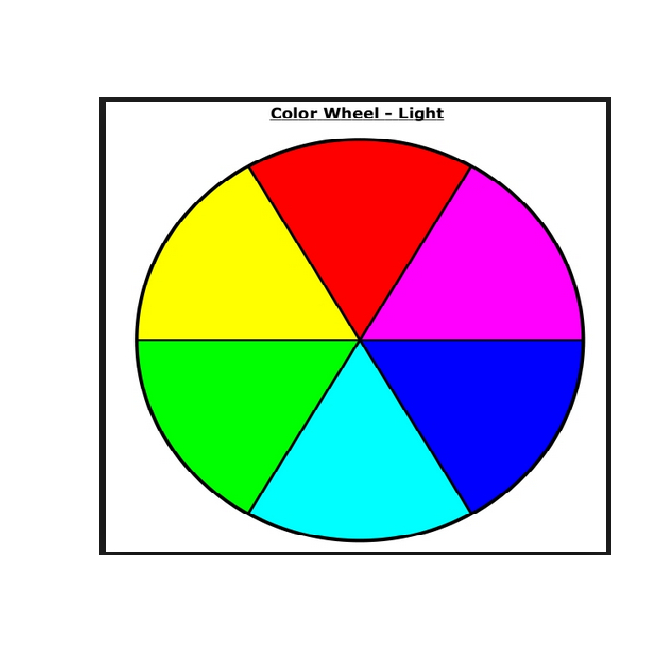
| in the second row returned from our call to pal(seq(0,1,len=6)). We see that 4 of the 6 numbers agree with our earlier call

| to pal. Two of the 6 differ slightly.

Not surprisingly the first color we see is FF0000, which we know represents red. The second color returned, FFFF00, must

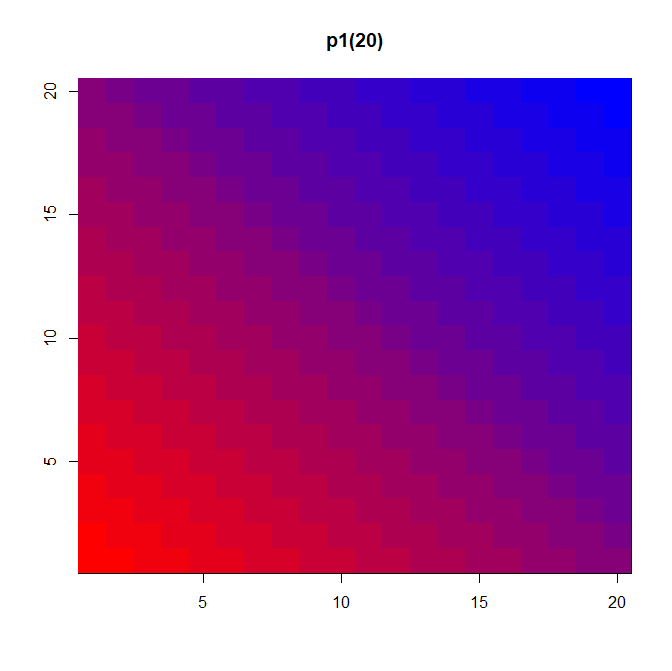
| represent yellow, a combination of full intensity red and full intensity green. This makes sense, since yellow falls

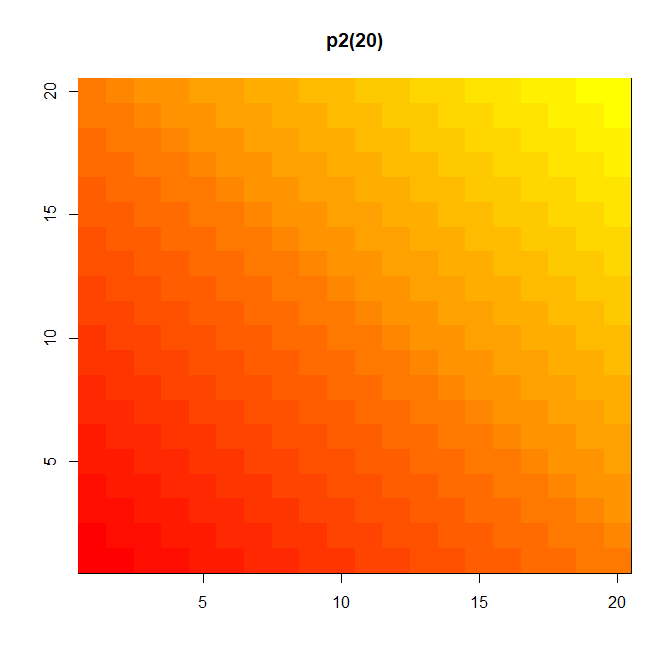
| between red and green on the color wheel as we see here. (We borrowed this image from lucaskrech.com.)



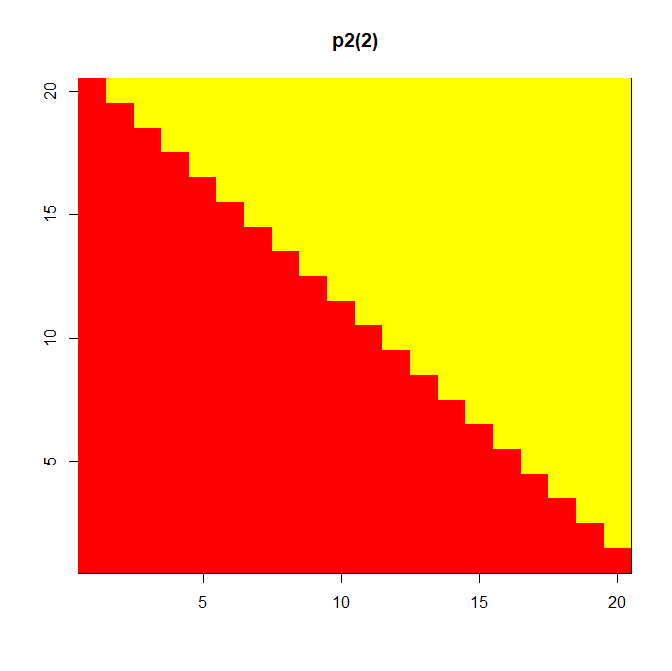
This is all fine and dandy…

showMe(p1(20)) and below is showMe(p2(20))





showMe(p2(2))



| A much more basic pattern, simple but elegant.

p1

function (n)

{

x <- ramp(seq.int(0, 1, length.out = n))

if (ncol(x) == 4L)

rgb(x[, 1L], x[, 2L], x[, 3L], x[, 4L], maxColorValue = 255)

else rgb(x[, 1L], x[, 2L], x[, 3L], maxColorValue = 255)

}

<bytecode: 0x0000000039f03268>

<environment: 0x0000000039f07dc8>

We see that p1 is a short function with one argument, n. The argument n is used as the length in a call to the function

| seq.int, itself an argument to the function ramp. We can infer that ramp is just going to divide the interval from 0 to 1

| into n pieces.

The heart of p1 is really the call to the function rgb with either 4 or 5 arguments. Use the ?fun construct to look at the R documentation for rgb now.

…

Well this picture is okay for a scatterplot, a nice mix of blue and green, but it really doesn't tell us too much

| information in the center portion, since the points are so thick there. We see there are a lot of points, but is one area

| more filled than another? We can't really discriminate between different point densities. This is where the alpha argument

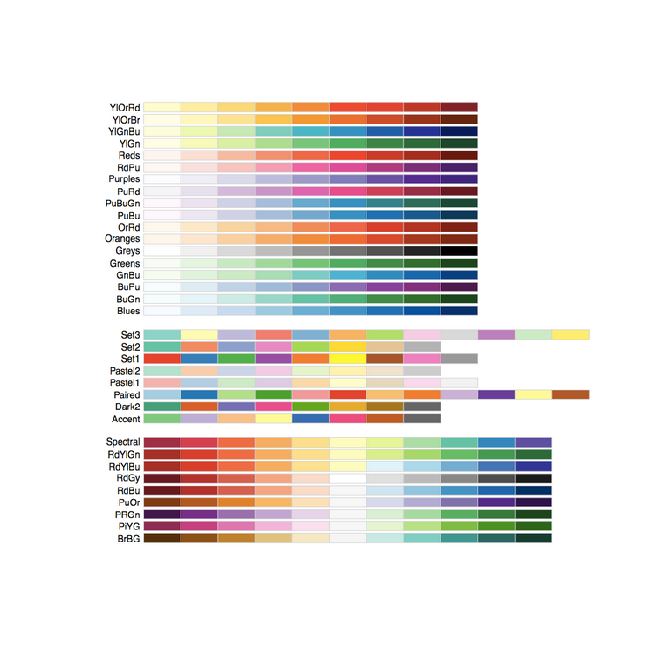
| can help us. Recall your plot command (use the up arrow) and add a 4th argument, .3, to the call to rgb. This will be our

| value for alpha.

> plot(x,y,pch=19,col=rgb(0,.5,.5,.3))

Clearly this is better. It shows us where, specifically, the densest areas of the scatterplot really are.

RColorBrewer Package, available on CRAN that contains interesting and useful color palettes, of which there are 3 types, sequential, divergent, and qualitative.



Here's a picture of the palettes available from this package. The top section shows the sequential palettes in which the

| colors are ordered from light to dark. The divergent palettes are at the bottom. Here the neutral color (white) is in the

| center, and as you move from the middle to the two ends of each palette, the colors increase in intensity. The middle

| display shows the qualitative palettes which look like collections of random colors. These might be used to distinguish

| factors in your data.

These colorBrewer palettes can be used in conjunction with the colorRamp() and colorRampPalette() functions. You would use

| colors from a colorBrewer palette as your base palette,i.e., as arguments to colorRamp or colorRampPalette which would

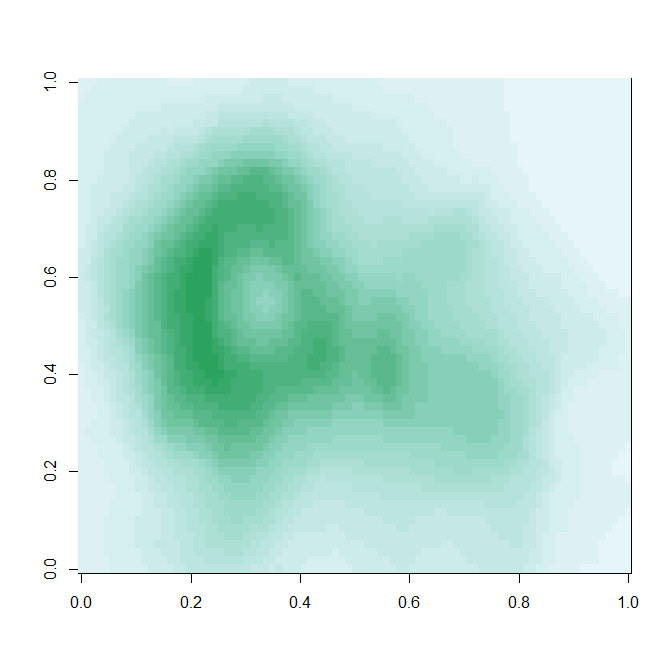
| interpolate them to create new colors.

| As an example of this, create a new object, cols by calling the function brewer.pal with 2 arguments, 3 and "BuGn". The

| string "BuGn" is the second last palette in the sequential display. The 3 tells the function how many different colors we

| want.

image(volcano,col=pal(20))



| Congratulations! We hope this lesson didn't make you see red. We're green with envy that you blue through it.

##use in lattice

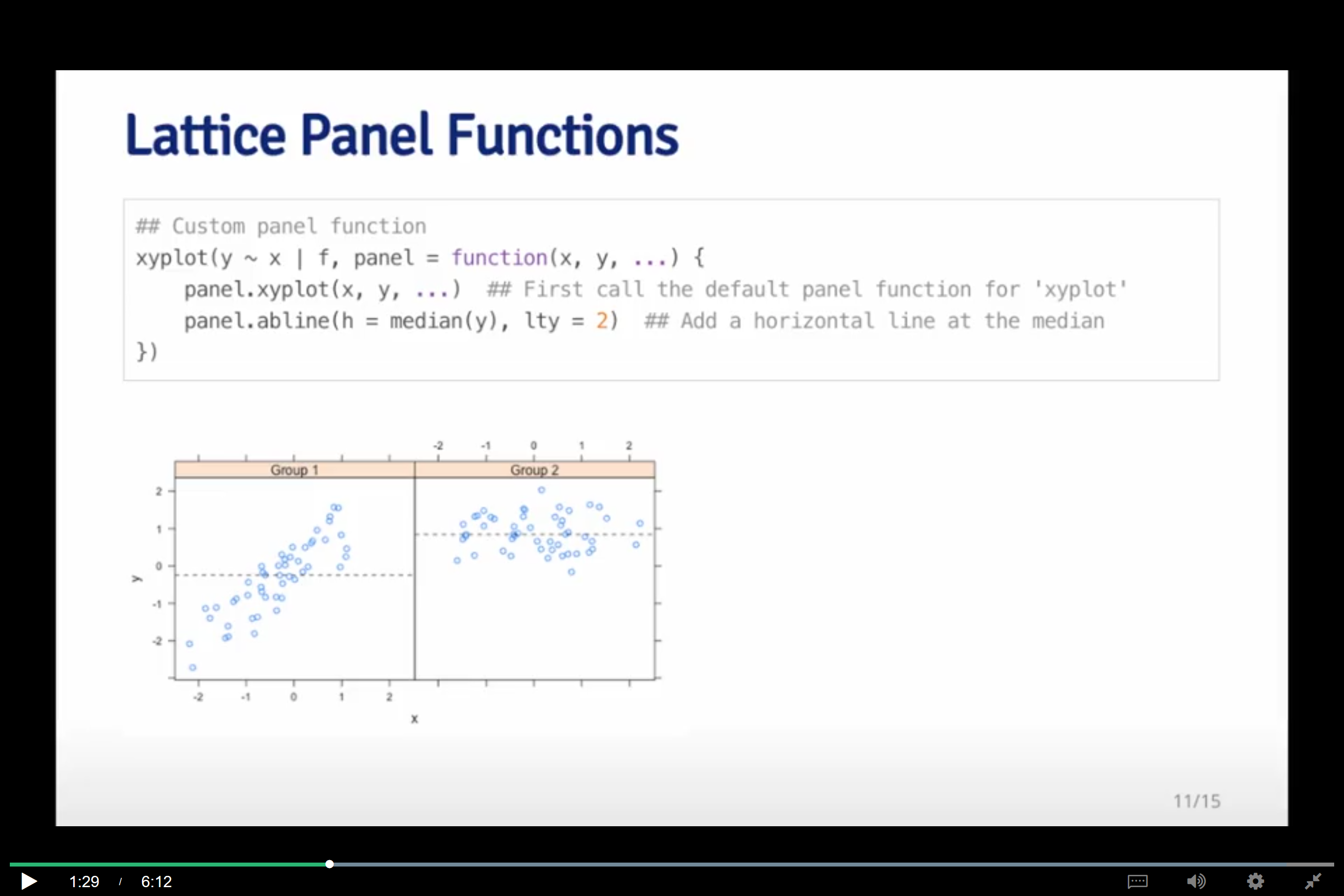
aa<-read.csv("aa.csv")

aa<-transform(aa,loctype=factor(loctype))

xyplot(OilConFctPS~date|loctype,aa,layout=c(6,1))

f<-factor(f,labels=c("Group1","Group2"))

xyplot(y~x|f,….

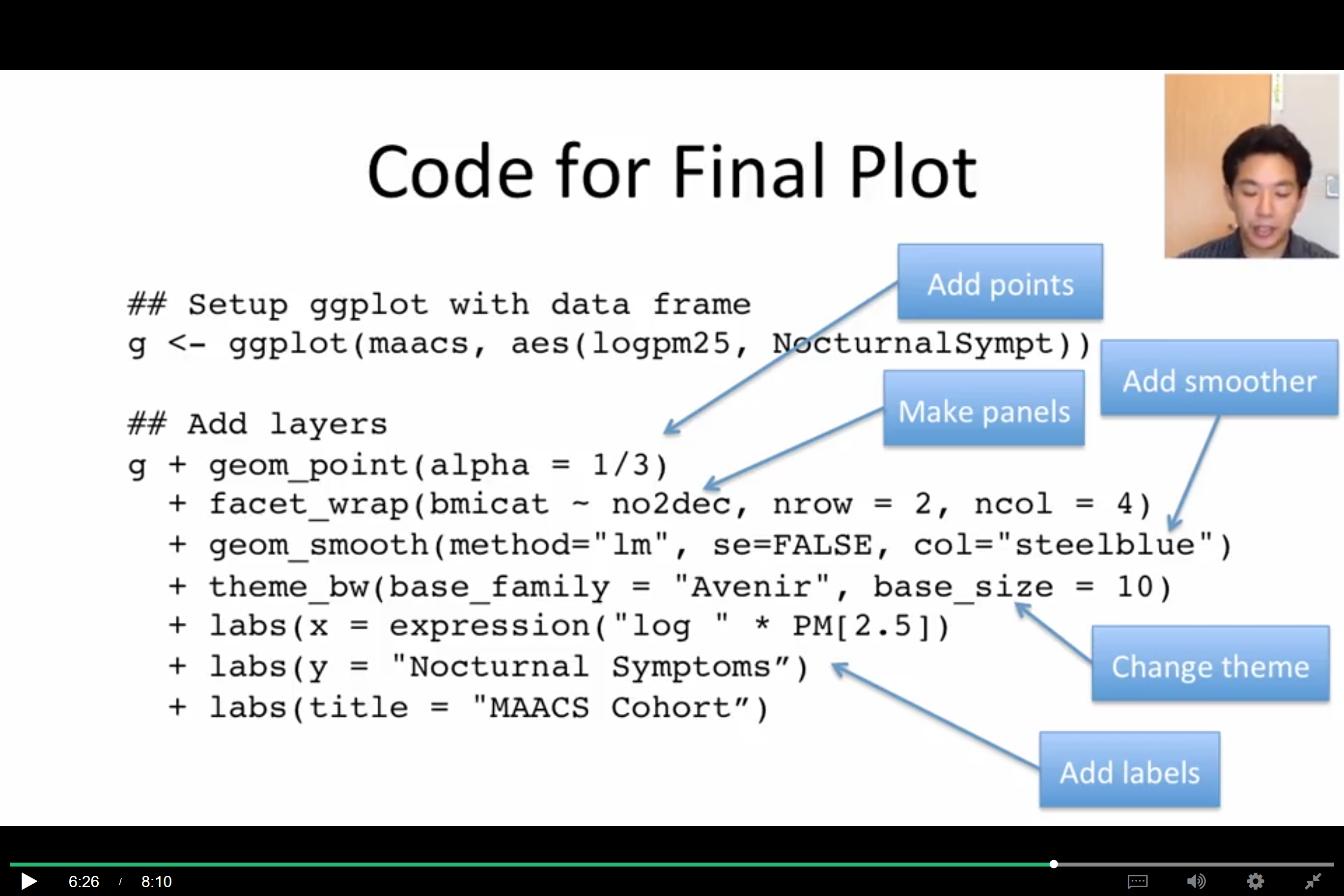




qplot(date,OilConFctPS,data=aa,color=loctype,geom=c("point","smooth")) ##con error

`geom\_smooth()` using method = 'loess' and formula 'y ~ x'

plot(filter(aa,location=="CAL")$date,filter(aa,location=="CAL")$OilConFctPS,type="l",ylab="Oil absorption factor",xlab="",xaxt="n",col="pink") ##ignora el pink, no pinta linea continua



str(ovw\_y2,list.len=ncol(ovw\_y2))

[diego.assencio.com](https://diego.assencio.com)

pause = function()

{

    if (interactive())

    {

        invisible(readline(prompt = "Press <Enter> to continue..."))

    }

    else

    {

        cat("Press <Enter> to continue...")

        invisible(readLines(file("stdin"), 1))

    }

}

swirl ggplot2

mpg dataset

diamond dataset

qplot(displ,hwy,data=mpg,color=drv,geom=c("point","smooth"))

qplot(y=hwy,data=mpg,color=drv)

specifying the y parameter only, without an x argument, plots the values of the y argument in the order in which they

occur in the data.

The all-purpose qplot can also create box and whisker plots. Call qplot now with 4 arguments. First specify the variable by which you'll split the data, in this case drv, then specify the variable which you want to examine, in this case hwy. The third argument is data (set equal to mpg), and the fourth, the geom, set equal to the string "boxplot"

> qplot(drv,hwy,data=mpg,geom="boxplot")

> qplot(drv,hwy,data=mpg,geom="boxplot",color=manufacturer)

Notice that there are still 3 regions of the plot (determined by the factor drv). Each is subdivided into several boxes depicting different manufacturers

> qplot(hwy,data=mpg,fill=drv)

Congrats! You've finished plot 1 of ggplot2. In the next lesson the plot thickens.

basic components of ggplot2 plots? There are 7 of them.

there's a DATA FRAME which contains the data you're trying to plot. Then the AESTHETIC MAPPINGS determine how data are mapped to color, size, etc. The GEOMS (geometric objects) are what you see in the plot (points, lines, shapes) and FACETS are the panels used in conditional plots.

There are 3 more. STATS are statistical transformations such as binning, quantiles, and smoothing which ggplot2 applies to the data. SCALES show what coding an aesthetic map uses (for example, male = red, female = blue). Finally, the plots are depicted on a COORDINATE SYSTEM. When you use qplot these were taken care of for you.

As in the base plotting system (and in contrast to the lattice system), when building plots with ggplot2, the plots are built up in layers, maybe in several steps. You can plot the data, then overlay a summary (for instance, a regression line or smoother) and then add any metadata and annotations you need.

g<-ggplot(mpg,aes(displ,hwy))

> summary(g)

data: manufacturer, model, displ, year, cyl, trans, drv, cty, hwy, fl, class [234x11]

mapping: x = ~displ, y = ~hwy

faceting: <ggproto object: Class FacetNull, Facet, gg>

compute\_layout: function

draw\_back: function

draw\_front: function

draw\_labels: function

draw\_panels: function

finish\_data: function

init\_scales: function

map\_data: function

params: list

setup\_data: function

setup\_params: function

shrink: TRUE

train\_scales: function

vars: function

super: <ggproto object: Class FacetNull, Facet, gg>

> g+geom\_point()+geom\_smooth()

`geom\_smooth()` using method = 'loess' and formula 'y ~ x'

The gray shadow around the blue line is the confidence band.

Now recall the

| expression you just used and add to it another call, this time to the function facet\_grid.

> g+geom\_point()+geom\_smooth(method="lm")+facet\_grid(.~drv)

You can add your own annotation using functions such as xlab(), ylab(), and ggtitle(). In addition, the function labs() is more general and can be used to label either or both axes as well as provide a title.

> g+geom\_point()+geom\_smooth(method="lm")+facet\_grid(.~drv)+ggtitle("Swirl Rules!")

g+geom\_point(color="pink",size=4,alpha=1/2)

Darker circles indicate values hit by multiple data points.

Note that you MUST use the function aes since the color of the points is data dependent and not a constant as it was in the previous example.

> g+geom\_point(size=4,alpha=1/2,aes(color=drv))

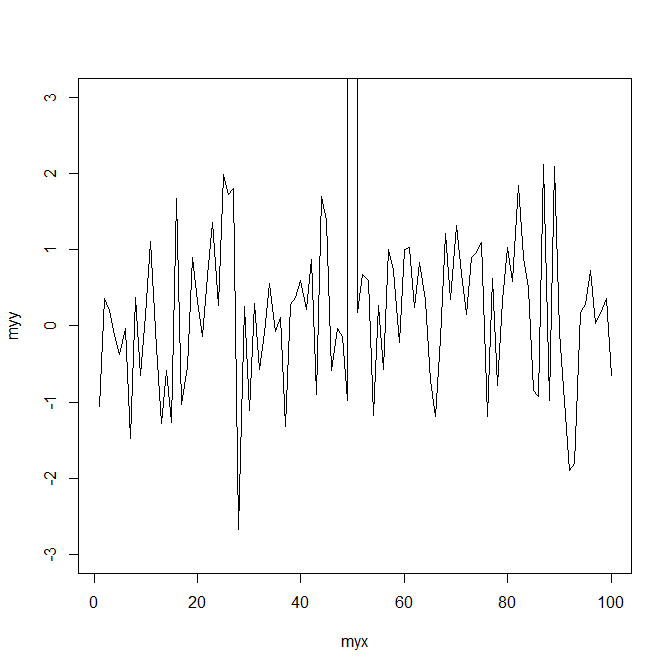
> g+geom\_point(aes(color=drv))+labs(title="Swirl Rules!")+labs(x="Displacement",y="Hwy Mileage")

> g+geom\_point(aes(color=drv),size=2,alpha=1/2)+geom\_smooth(size=4,linetype=3,method="lm",se=FALSE)

gray shadows indicating standard errors (confidence intervals)

g+geom\_point(aes(color=drv))+theme\_bw(base\_family="Times")

plot(myx,myy,type="l",ylim=c(-3,3))

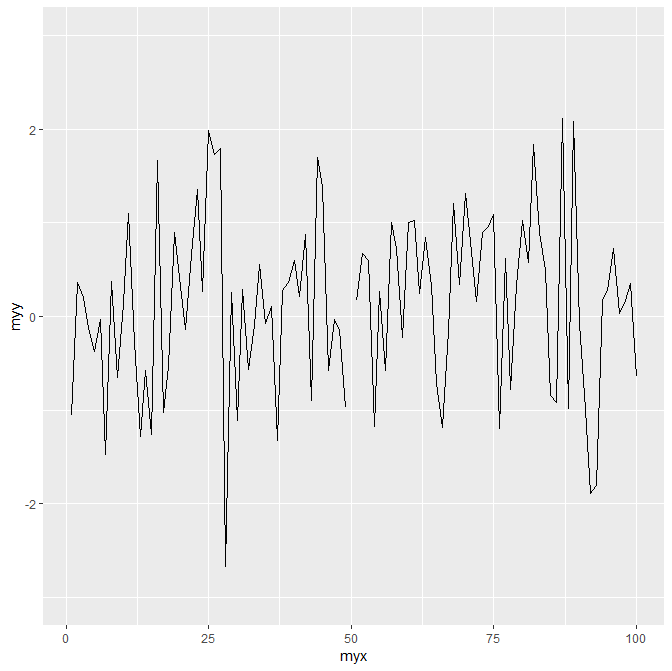


g<-ggplot(testdat,aes(x=myx,y=myy))

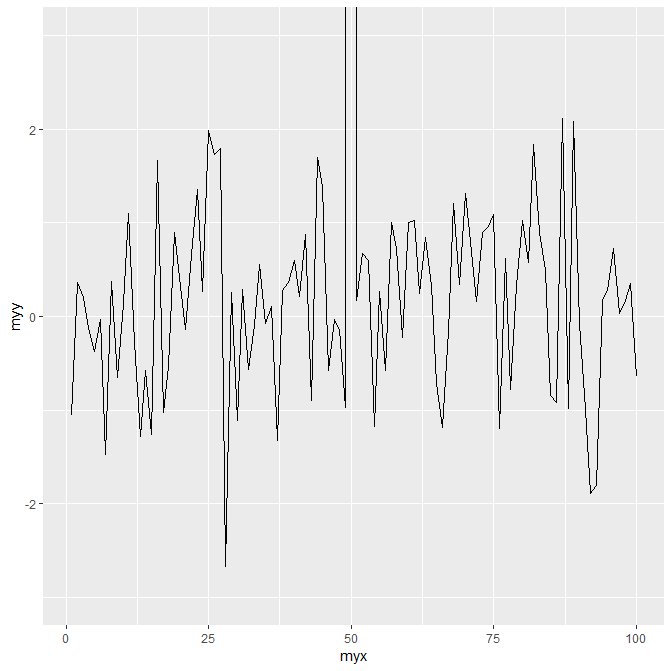
> g+geom\_line()



g+geom\_line()+ylim(-3,3)

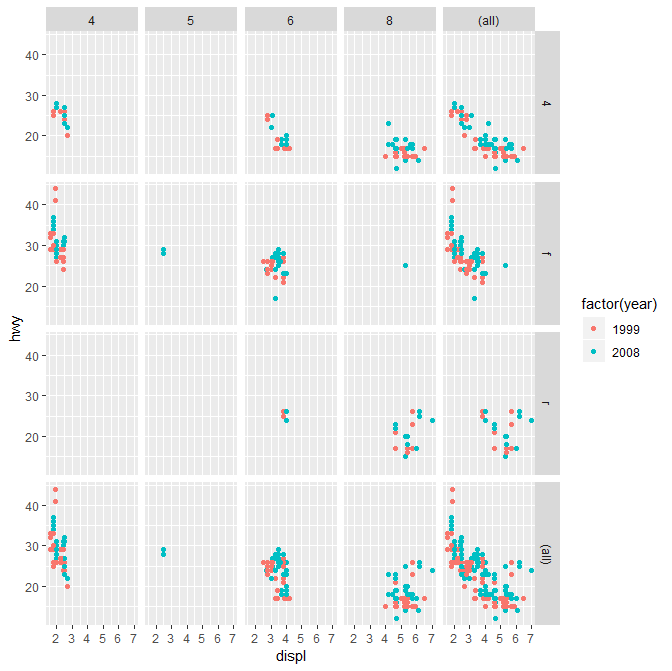


g+geom\_line()+coord\_cartesian(ylim=c(-3,3))



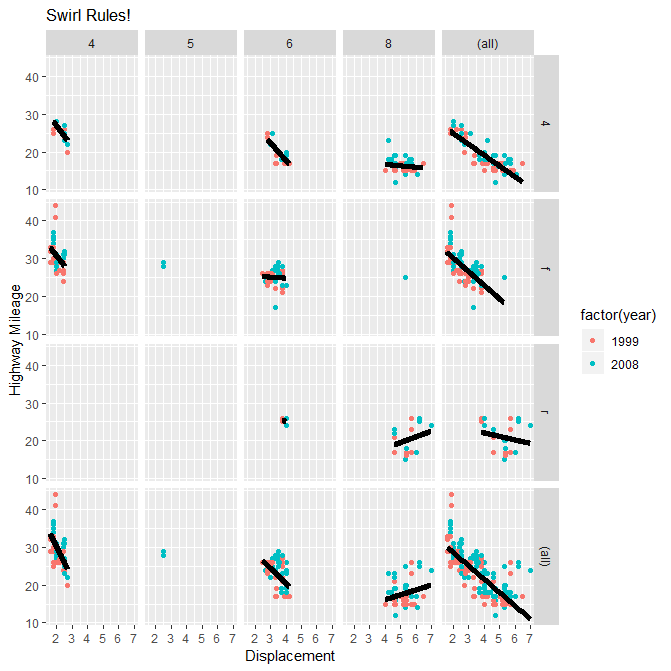
g<-ggplot(mpg,aes(x=displ,y=hwy,color=factor(year)))

g+geom\_point()+facet\_grid(drv~cyl,margins=TRUE)



The margins argument tells ggplot to display the marginal totals over each row and column, so instead of seeing 3 rows (the number of drv factors) and 4 columns (the number of cyl factors) we see a 4 by 5 display.

> g+geom\_point()+facet\_grid(drv~cyl,margins=TRUE)+geom\_smooth(method="lm",se=FALSE,size=2,color="black")+labs(x="Displacement",y="Highway Mileage",title="Swirl Rules!")



Histogram

> qplot(price,data=diamonds,geom="density")

> qplot(price,data=diamonds,geom="density",color=cut)

Scatterplot

qplot(carat,price,data=diamonds)

qplot(carat,price,data=diamonds,shape=cut)

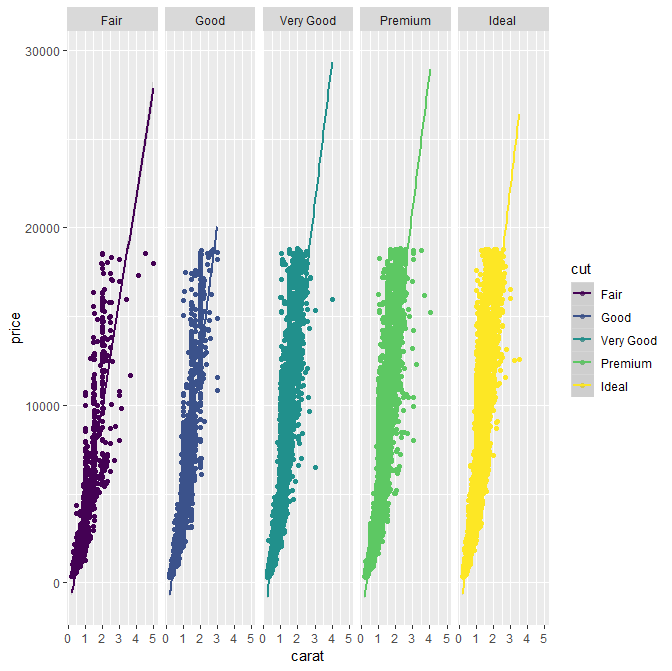
Warning message: Using shapes for an ordinal variable is not advised

qplot(carat,price,data=diamonds,color=cut)

qplot(carat,price,data=diamonds, color=cut) + geom\_smooth(method="lm")

the better the cut, the steeper (more positive) the slope of the lines.

qplot(carat,price,data=diamonds,color=cut,facets=.~cut)+geom\_smooth(method="lm")



g<-ggplot(diamonds,aes(depth,price))

> summary(g)

data: carat, cut, color, clarity, depth, table, price, x, y, z [53940x10]

mapping: x = ~depth, y = ~price

faceting: <ggproto object: Class FacetNull, Facet, gg>

compute\_layout: function

draw\_back: function

draw\_front: function

draw\_labels: function

draw\_panels: function

finish\_data: function

init\_scales: function

map\_data: function

params: list

setup\_data: function

setup\_params: function

shrink: TRUE

train\_scales: function

vars: function

super: <ggproto object: Class FacetNull, Facet, gg>

We see that g holds the entire dataset

the boolean na.rm set equal to TRUE.

cutpoints<-quantile(diamonds$carat,seq(0,1,length=4),na.rm=TRUE)

cutpoints

0% 33.33333% 66.66667% 100%

0.20 0.50 1.00 5.01

We see a 4-long vector (explaining why length was set equal to 4). We also see that .2 is the smallest carat size in the dataset and 5.01 is the largest. One third of the diamonds are between .2 and .5 carats and another third are between .5 and 1 carat in size. The remaining third are between 1 and 5.01 carats. Now we can use the R command cut to label each of the 53940 diamonds in the dataset as belonging to one of these 3 factors. Create a new name in diamonds, diamonds$car2 by assigning it the output of the call to cut. This command takes 2 arguments, diamonds$carat, which is what we want to cut, and cutpoints, the places where we'll cut.

diamonds$car2<-cut(diamonds$carat,cutpoints)

Now we can continue with our multi-facet plot. First we have to reset g since we changed the dataset (diamonds) it contained (by adding a new column). Assign to g the output of a call to ggplot with 2 arguments. The dataset diamonds is the first, and a call to the function aes with 2 arguments (depth,price) is the second.

> g<-ggplot(diamonds,aes(depth,price))

> g+geom\_point(alpha=1/3)+facet\_grid(cut~car2)

We created a vector, myd, containing the indices of these datapoints. Look at these entries in diamonds by typing the expression diamonds[myd,]. The myd tells R what rows to show and the empty column entry says to print all the columns.

> diamonds[myd,]

# A tibble: 12 x 11

carat cut color clarity depth table price x y z car2

<dbl> <ord> <ord> <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <fct>

1 0.2 Premium E SI2 60.2 62 345 3.79 3.75 2.27 <NA>

2 0.2 Premium E VS2 59.8 62 367 3.79 3.77 2.26 <NA>

3 0.2 Premium E VS2 59 60 367 3.81 3.78 2.24 <NA>

4 0.2 Premium E VS2 61.1 59 367 3.81 3.78 2.32 <NA>

5 0.2 Premium E VS2 59.7 62 367 3.84 3.8 2.28 <NA>

6 0.2 Ideal E VS2 59.7 55 367 3.86 3.84 2.3 <NA>

7 0.2 Premium F VS2 62.6 59 367 3.73 3.71 2.33 <NA>

8 0.2 Ideal D VS2 61.5 57 367 3.81 3.77 2.33 <NA>

9 0.2 Very Good E VS2 63.4 59 367 3.74 3.71 2.36 <NA>

10 0.2 Ideal E VS2 62.2 57 367 3.76 3.73 2.33 <NA>

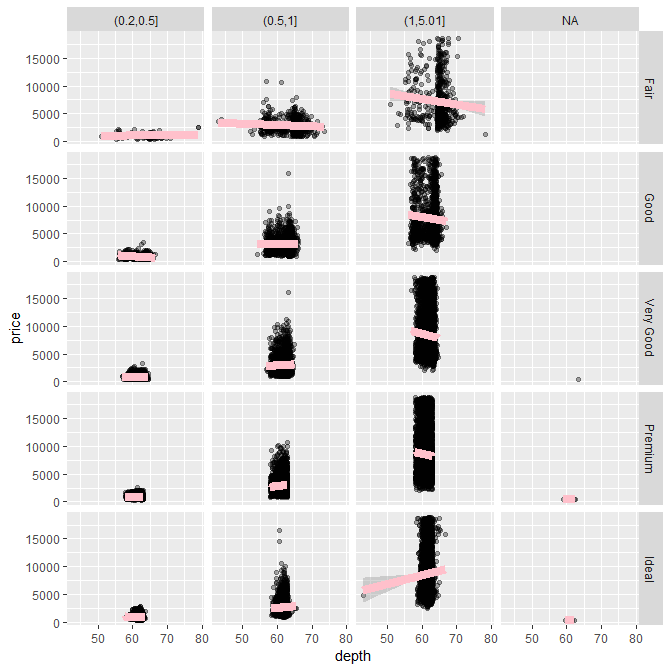
11 0.2 Premium D VS2 62.3 60 367 3.73 3.68 2.31 <NA>

12 0.2 Premium D VS2 61.7 60 367 3.77 3.72 2.31 <NA>

We see these entries match the plots. Whew - that's a relief. The car2 field is, in fact, NA for these entries, but the carat field shows they each had a carat size of .2. What's going on here?

Actually our plot answers this question. The boundaries for each column appear in the gray labels at the top of each column, and we see that the first column is labeled (0.2,0.5]. This indicates that this column contains data greater than .2 and less than or equal to .5. So diamonds with carat size .2 were excluded from the car2 field.

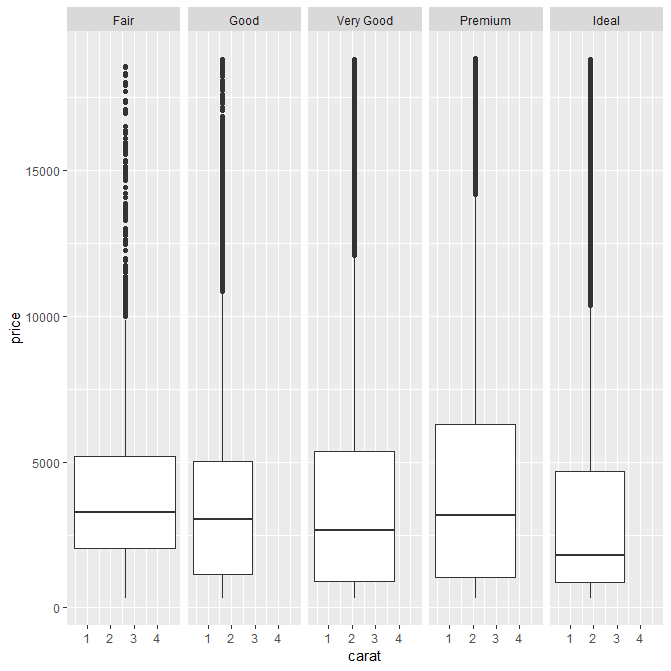
> g+geom\_point(alpha=1/3)+facet\_grid(cut~car2)+geom\_smooth(method="lm",size=3,color=pink)



> ggplot(diamonds,aes(carat,price))+geom\_boxplot()+facet\_grid(.~cut)

Warning message:

Continuous x aesthetic -- did you forget aes(group=...)?



Yes! A boxplot looking like marshmallows about to be roasted. Well done and congratulations! You've finished this jewel of a lesson. Hope it paid off!

From quiz week 2, course 4, exploratory data analysis

> qplot(Wind, Ozone, data = airquality, facets = . ~ factor(Month))

or

> airquality = transform(airquality, Month = factor(Month))

> qplot(Wind, Ozone, data = airquality, facets = . ~ Month)

> qplot(votes, rating, data = movies) + geom\_smooth()

`geom\_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'

**Congratulations! You passed!**

Question 1

*Correct*

1 / 1

point

## 1. Question 1

Under the lattice graphics system, what do the primary plotting functions like xyplot() and bwplot() return?



an object of class "lattice"



nothing; only a plot is made



an object of class "plot"



an object of class "trellis"

Correct

Question 2

*Correct*

1 / 1

point

## 2. Question 2

What is produced by the following code?





A set of 16 panels showing the relationship between weight and time for each rat.



A set of 3 panels showing the relationship between weight and time for each rat.



A set of 3 panels showing the relationship between weight and time for each diet.

Correct



A set of 11 panels showing the relationship between weight and diet for each time.

Question 3

*Correct*

1 / 1

point

## 3. Question 3

Annotation of plots in any plotting system involves adding points, lines, or text to the plot, in addition to customizing axis labels or adding titles. Different plotting systems have different sets of functions for annotating plots in this way.

Which of the following functions can be used to annotate the panels in a multi-panel lattice plot?



points()



panel.abline()

Correct



text()



lines()



axis()

Question 4

*Correct*

1 / 1

point

## 4. Question 4

The following code does NOT result in a plot appearing on the screen device.



Which of the following is an explanation for why no plot appears?



The object 'p' has not yet been printed with the appropriate print method.

Correct



The xyplot() function, by default, sends plots to the PDF device.



There is a syntax error in the call to xyplot().



The variables being plotted are not found in that dataset.

Question 5

*Correct*

1 / 1

point

## 5. Question 5

In the lattice system, which of the following functions can be used to finely control the appearance of all lattice plots?



splom()



trellis.par.set()

Correct



par()



print.trellis()

Question 6

*Correct*

1 / 1

point

## 6. Question 6

What is ggplot2 an implementation of?



the Grammar of Graphics developed by Leland Wilkinson

Correct



the base plotting system in R



a 3D visualization system



the S language originally developed by Bell Labs

Question 7

*Correct*

1 / 1

point

## 7. Question 7

Load the `airquality' dataset form the datasets package in R



I am interested in examining how the relationship between ozone and wind speed varies across each month. What would be the appropriate code to visualize that using ggplot2?













Correct





Question 8

*Correct*

1 / 1

point

## 8. Question 8

What is a **geom** in the ggplot2 system?



a method for making conditioning plots



a plotting object like point, line, or other shape

Correct



a method for mapping data to attributes like color and size



a statistical transformation

Question 9

*Correct*

1 / 1

point

## 9. Question 9

When I run the following code I get an error:



I was expecting a scatterplot of 'votes' and 'rating' to appear. What's the problem?



There is a syntax error in the call to ggplot.



The dataset is too large and hence cannot be plotted to the screen.



ggplot does not yet know what type of layer to add to the plot.

Correct



The object 'g' does not have a print method.

Question 10

*Correct*

1 / 1

point

## 10. Question 10

The following code creates a scatterplot of 'votes' and 'rating' from the movies dataset in the ggplot2 package. After loading the ggplot2 package with the library() function, I can run



How can I modify the the code above to add a smoother to the scatterplot?

















Correct

In R, one can write a conditional statement as follows:

ifelse(condition on data, true value returned, false returned)

The above expression reads: if condition on the data is true, then do the true value assigned; otherwise execute the "false value."

> ifelse(3 > 4, x <- 5, x <- 6)

> x

[1] 6

**forcali<-ecdf(subset(yp\_fg18s,location=="CAL")$psy)**

**> forcali(1.174)**

**[1] 0**

**> forcali(1.374)**

**[1] 0.2826087**

pts <- pretty(x / 10000000)

axis(2, at = pts, labels = paste(pts, "MM", sep = ""))

format number

formatC(1000.64, format="f", big.mark=",", digits=1)

gridlines

For reference, there is a way to control the grid and axes parameters directly from the plot() command, if we are not defining a custom tick interval:

plot(x = 1:10, y = rnorm(10, 5, 2), xlim=c(1, 10), ylim=c(1, 10), panel.first=grid())

The plot.default() documentation gives more information about these parameters.

When using a custom ticks interval, the easiest is to draw the grid using abline:

plot(x = 1:10, y = rnorm(10, 5, 2), xaxp=c(1, 10, 10), yaxp=c(1, 10, 10), axes=FALSE)

axis(1, 1:10)

axis(2, 1:10)

abline(h=1:10, v=1:10, col="gray", lty=3)

**YP\_R**

ypfg<-function()**{**

yp\_fg<-read.csv("2 2) Fior all data.csv")

yp\_p<-read.csv("2 2) PELLETS all data.csv")

##proper format for dates

yp\_fg<-mutate(yp\_fg,date=as.Date(as.character(Date),"%d-%b-%y"), weekc=as.Date(as.character(WeekC),"%d-%b-%y"),location=as.character(Location))

yp\_p<-mutate(yp\_p,date=as.Date(as.character(Date),"%d-%b-%y"), weekc=as.Date(as.character(WeekC),"%d-%b-%y"),location=as.character(Location))

yp\_p17<-filter(arrange(yp\_p,location,desc(weekc)),weekc<"2017-12-31"&weekc>="2017-09-01")

yp\_fg17<-filter(arrange(yp\_fg,location,desc(weekc)),weekc<"2017-12-31"&weekc>="2017-09-01")

##calculate 2017 averages

yp\_fg17s<-summarize(group\_by(select(yp\_fg17,location,weekc,PS.Actual.FG.LBS,PS.Fried.Lbs,CR.TT.Finished.Goods.Lbs,CR.TT.Fried.Lbs),location,weekc),psac=sum(PS.Actual.FG.LBS,na.rm=TRUE),psfr=sum(PS.Fried.Lbs,na.rm=TRUE),crac=sum(CR.TT.Finished.Goods.Lbs,na.rm=TRUE),crfr=sum(CR.TT.Fried.Lbs,na.rm=TRUE),psy=psac/psfr,cry=crac/crfr)

yp\_fg17m<-summarize(group\_by(yp\_fg17s,location),psac17=sum(psac),psfr17=sum(psfr),crac17=sum(crac),crfr17=sum(crfr),psy17=mean(psac17/psfr17),cry17=mean(crac17/crfr17))

##why not trying to calculate mean of weekly yields with a loop to avoid NaR…

*##yp\_p17s<-summarize(group\_by(select(yp\_p17,Location,weekc,………………..PS.Actual.FG.LBS,PS.Fried.Lbs,CR.TT.Finished.Goods.Lbs,CR.TT.Fried.Lbs),Location,WeekC),psac=sum(PS.Actual.FG.LBS,na.rm=TRUE),psfr=sum(PS.Fried.Lbs,na.rm=TRUE),crac=sum(CR.TT.Finished.Goods.Lbs,na.rm=TRUE),crfr=sum(CR.TT.Fried.Lbs,na.rm=TRUE))*

##first build the 2018 weekly yields data frame

yp\_fg18<-filter(arrange(yp\_fg,location,desc(weekc)),weekc<="2018-12-31"&weekc>="2017-12-31")

yp\_p18<-filter(arrange(yp\_p,location,desc(weekc)),weekc<="2018-12-31"&weekc>="2017-12-31")

yp\_fg18s<-summarize(group\_by(select(yp\_fg18,location,weekc,PS.Actual.FG.LBS,PS.Fried.Lbs,CR.TT.Finished.Goods.Lbs,CR.TT.Fried.Lbs),location,weekc),psac=sum(PS.Actual.FG.LBS,na.rm=TRUE),psfr=sum(PS.Fried.Lbs,na.rm=TRUE),crac=sum(CR.TT.Finished.Goods.Lbs,na.rm=TRUE),crfr=sum(CR.TT.Fried.Lbs,na.rm=TRUE),psy=ifelse(psac==0,NA,psac/psfr),cry=ifelse(crac==0,NA,crac/crfr))

yp\_fg18m<-summarize(group\_by(yp\_fg18s,location),psac18=sum(psac),psfr18=sum(psfr),crac18=sum(crac),crfr18=sum(crfr),psy18=mean(psac18/psfr18),cry18=mean(crac18/crfr18))

coiy18<-subset(yp\_fg18s,location=="COI",select=c(location,weekc,psy,cry))

##yp\_fg17m[6,6]<-min(coiy18$psy)

##yp\_fg17m[6,7]<-min(coiy18$cry)

yp\_fg17m[6,6]<-quantile((coiy18$psy),probs=0.33)

yp\_fg17m[6,7]<-quantile((coiy18$cry),probs=0.33)

yp\_fg17m[6,1]<-"COI"

pssav<-numeric()

crsav<-numeric()

k<-0

m<-0

for(i in 1:nrow(yp\_fg18s))**{**

pssav[i]=k+1.26\*(-as.numeric(yp\_fg18s[i,4])+as.numeric(yp\_fg18s[i,3])/ as.numeric(filter(yp\_fg17m,location==as.character(yp\_fg18s[i,1]))[,6]))

crsav[i]= m+1.26\*(-as.numeric(yp\_fg18s[i,6])+as.numeric(yp\_fg18s[i,5])/ as.numeric(filter(yp\_fg17m,location==as.character(yp\_fg18s[i,1]))[,7]))

ifelse(as.character(yp\_fg18s[i+1,1])== as.character(yp\_fg18s[i,1]),k<-pssav[i],k<-0)

ifelse(as.character(yp\_fg18s[i+1,1])== as.character(yp\_fg18s[i,1]),m<-crsav[i],m<-0)

**}**

sav18<-data.frame(yp\_fg18s$location, yp\_fg18s$weekc, yp\_fg18s$psy ,pssav, yp\_fg18s$cry ,crsav)

evsav18<-summarize(group\_by(subset(sav18, yp\_fg18s.location!="COI"),yp\_fg18s.weekc),sum(pssav),sum(crsav,na.rm=TRUE))

evsav18<-mutate(evsav18,location="Evans")

names(evsav18)<-c("weekc","pssav","crsav","location")

write.csv(sav18,"sav18.csv")

write.csv(yp\_fg18m ,"yp\_fg18m.csv")

write.csv(yp\_fg17m,"yp\_fg17m.csv")

write.csv(evsav18,"evsav18.csv")

plot.new()

dev.new()

ifelse(min(evsav18$pssav,sav18$pssav,na.rm=TRUE)>0,y1<-0,y1<- min(evsav18$pssav,sav18$pssav,na.rm=TRUE))

ifelse(max(evsav18$pssav,sav18$pssav,na.rm=TRUE)<0,y2<-0,y2<-max(evsav18$pssav,sav18$pssav,na.rm=TRUE))

plot(filter(sav18,yp\_fg18s.location=="CAL")$yp\_fg18s.weekc,filter(sav18,yp\_fg18s.location=="CAL")$pssav,type="l",col="pink",lwd=2,xlab="weeks of 2018",ylab="Accumulated savings (k usd)",main="FG PS yield savings 2018 vs 2017", ylim=c(y1,y2),yaxt="n",bty="n")

points(filter(sav18,yp\_fg18s.location=="TX")$yp\_fg18s.weekc,filter(sav18,yp\_fg18s.location=="TX")$pssav,col="green",type="l",lwd=2, bty="n")

pts <-seq(y1,y2,by=(y2-y1)/3) ## c(100000,200000,300000,400000)

ptsv<-seq(0,length(unique(evsav18$weekc)),by=1)

axis**(**2, at = c**(**0,as.numeric**(**evsav18[length(evsav18$pssav),2]**)** ,pts**)**, labels = c(0,formatC**(**as.numeric**(**evsav18[length**(**evsav18$pssav**)**,2]/1000**)**,format="f",digits=0,big.mark=","**)**,paste**(**formatC**(**pts/1000,format="f",digits=0,big.mark=","**)**, "k", sep = ""**)**),las=1**)**

points(filter(sav18,yp\_fg18s.location=="OH")$yp\_fg18s.weekc,filter(sav18,yp\_fg18s.location=="OH")$pssav,col="blue",type="l",lwd=2,bty="n")

points(filter(sav18,yp\_fg18s.location=="CHI")$yp\_fg18s.weekc,filter(sav18,yp\_fg18s.location=="CHI")$pssav,col="orange",type="l",lwd=2,bty="n")

points(filter(sav18,yp\_fg18s.location=="COI")$yp\_fg18s.weekc,filter(sav18,yp\_fg18s.location=="COI")$pssav,col="red",type="l",lwd=2,bty="n")

points(filter(sav18,yp\_fg18s.location=="MEX")$yp\_fg18s.weekc,filter(sav18,yp\_fg18s.location=="MEX")$pssav,col="purple",type="l",lwd=2,bty="n")

points(evsav18$weekc,evsav18$pssav,col="brown",type="l",lwd=2,bty="n")

legend("topleft",lty=1:1,col=c("pink","green","blue","orange","red","purple","brown"),legend=c("CAL","TX","OH","CHI","COI","MEX","Evans"),ncol=2,lwd=3)

##abline(h=pts,v=numeric(unique(month(evsav18$weekc,label=TRUE,abbr=TRUE))),col=" lightgray ", lwd=1,lty="dotted")

##abline(h=pts,v=(unique(month(evsav18$weekc))),col=" lightgray ", lwd=1,lty="dotted")

abline(h=pts,v=ptsv,col=" lightgray", lwd=1,lty="dotted")

abline(h=0,col=" lightgray", lwd=1,lty="dotted")

abline(h= as.numeric(evsav18[length(evsav18$pssav),2]),col=" lightgray", lwd=1,lty="dotted")

pause = function()

{

    if (interactive())

    {

        invisible(readline(prompt = " Please copy the graph, click on this window and press <Enter> to continue..."))

    }

    else

    {

        cat("Please copy the graph, click on this window and press <Enter> to continue...")

        invisible(readLines(file("stdin"), 1))

    }

}

pause()

plot.new()

dev.new()

ifelse(min(evsav18$crsav,sav18$crsav,na.rm=TRUE)>0,y1<-0,y1<- min(evsav18$crsav,sav18$crsav,na.rm=TRUE))

ifelse(max(evsav18$crsav,sav18$crsav,na.rm=TRUE)<0,y2<-0,y2<- max(evsav18$crsav,sav18$crsav,na.rm=TRUE))

plot(filter(sav18,yp\_fg18s.location=="CAL")$yp\_fg18s.weekc,filter(sav18,yp\_fg18s.location=="CAL")$crsav,type="l",col="pink",lwd=2,xlab="weeks of 2018",ylab="Accumulated savings (k usd)",main="FG CR yield savings 2018 vs 2017", ylim=c(y1,y2),yaxt="n",bty="n")

points(filter(sav18,yp\_fg18s.location=="TX")$yp\_fg18s.weekc,filter(sav18,yp\_fg18s.location=="TX")$crsav,col="green",type="l",lwd=2, bty="n")

pts <-seq(y1,y2,by=(y2-y1)/3) ## c(-25000,50000,100000,150000,200000)

ptsv<-seq(0,length(unique(evsav18$weekc)),by=1)

##axis(2, at = c(0,pts), labels = c(0,paste(formatC(pts/1000,format="f",digits=0,big.mark=","), "k", sep = "")),las=1)

axis**(**2, at = c**(**as.numeric**(**evsav18[length(evsav18$crsav),3]**)**,0,pts**)**, labels = c(formatC**(**as.numeric**(**evsav18[length**(**evsav18$crsav**)**,3]/1000**)**,format="f",digits=0,big.mark=","**)**,0,paste**(**formatC**(**pts/1000,format="f",digits=0,big.mark=","**)**, "k", sep = ""**)**),las=1**)**

##axis(2, at = c(0,pts,as.numeric(evsav18[length(evsav18$crsav),3])), labels = c(0,paste(formatC(pts/1000,format="f",digits=0,big.mark=","), "k", sep = ""), formatC(as.numeric(evsav18[length(evsav18$crsav),3]),format="f",digits=0,big.mark=",")))

points(filter(sav18,yp\_fg18s.location=="OH")$yp\_fg18s.weekc,filter(sav18,yp\_fg18s.location=="OH")$crsav,col="blue",type="l",lwd=2,bty="n")

points(filter(sav18,yp\_fg18s.location=="CHI")$yp\_fg18s.weekc,filter(sav18,yp\_fg18s.location=="CHI")$crsav,col="orange",type="l",lwd=2,bty="n")

points(filter(sav18,yp\_fg18s.location=="COI")$yp\_fg18s.weekc,filter(sav18,yp\_fg18s.location=="COI")$crsav,col="red",type="l",lwd=2,bty="n")

points(filter(sav18,yp\_fg18s.location=="MEX")$yp\_fg18s.weekc,filter(sav18,yp\_fg18s.location=="MEX")$crsav,col="purple",type="l",lwd=2,bty="n")

points(evsav18$weekc,evsav18$crsav,col="brown",type="l",lwd=2,bty="n")

legend("bottomleft",lty=1:1,col=c("pink","green","blue","red","brown"),legend=c("CAL","TX","OH","COI","Evans"),ncol=2,lwd=3)

abline(h=pts,v=ptsv,col=" lightgray", lwd=1,lty="dotted")

abline(h=0,col=" lightgray", lwd=1,lty="dotted")

abline(h= as.numeric(evsav18[length(evsav18$crsav),3]),col=" lightgray", lwd=1,lty="dotted")

**}**

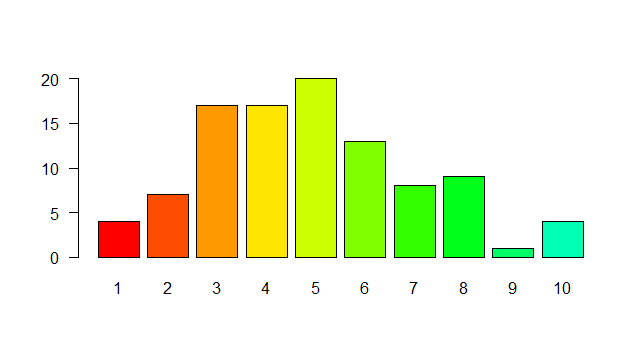
cepte

Not sure if this is what you mean, but try setting las=1. Here's an example:

require(grDevices)

tN <- table(Ni <- stats::rpois(100, lambda=5))

r <- barplot(tN, col=rainbow(20), las=1)



“Shorten the distance from mind to page”.

Center the ggtitle:

ggtitle("Average bill for 2 people") +

theme\_bw() +

**to change axis labels format**

**+theme(axis.text.x = element\_text(face="bold", color="#993333", size=9, angle=45), axis.text.y = element\_text(face="bold", color="#993333", size=9, angle=45))+ scale\_y\_continuous(labels = scientific)**

Factors as characters…

Factors are structured as numeric indices tied to a list of 'levels'. This can be seen if you convert a factor to numeric. So:

> fact <- as.factor(c("a","b","a","d")

> fact

[1] a b a d

Levels: a b d

> as.numeric(fact)

[1] 1 2 1 3

**The numbers returned in the last line correspond to the levels of the factor.**

> levels(fact)

[1] "a" "b" "d"

Notice that levels() returns an array of characters. You can use this fact to easily and compactly convert factors to strings or numerics like this:

> fact\_character <- levels(fact)[as.numeric(fact)]

> fact\_character

[1] "a" "b" "a" "d"

class(yield17$fet)

[1] "factor"

> levels(yield17$fet)

[1] "CR" "CR Din" "CR Ten" "M-6" "M-7"

[6] "M-7 Fz" "M-8 GF" "M5 Ham" "M5B" "M5B Fz"

[11] "S-2" "S-2 Ga" "S-2 Plate" "S-3" "ST"

[16] "ST 1x3" "ST 1x5" "ST 1x9" "ST 3/4" "ST B"

[21] "ST HB" "ST L" "ST S" "ST Shoulder" "Str. Long / Short & 1x9"

[26] "STRIPS ABF"

> levels(yield17$fet)[as.numeric(yield17$fet[3])]

[1] "CR"

> class(levels(yield17$fet)[as.numeric(yield17$fet[3])])

[1] "character"

newbob <- transform(bob, phenotype = as.character(phenotype))

Just be sure to put every factor you'd like to convert to character.

Or you can do something like this and kill all the pests with one blow:

newbob\_char <- as.data.frame(lapply(bob[sapply(bob, is.factor)], as.character), stringsAsFactors = FALSE)

newbob\_rest <- bob[!(sapply(bob, is.factor))]

newbob <- cbind(newbob\_char, newbob\_rest)

All weeks in 2018:

days2018<-seq(from=as.Date("2018-01-01"),to=as.Date("2018-12-31"),by =1)

days2018\_d<-transform(as.Date(days2018))

weeks2018<-mutate(days2018\_d,weekc=days2018\_d[,1]+1-wday(days2018\_d[,1]))

weeks18<-unique(weeks2018$weekc)

subset/filter data frame with variables for two conditions on columns:

yw\_t<-yw18[grep(yield$fet[j],yw18$fet),]

yw\_tw<-yw\_t[grep(weeks18u[i],yw\_t$weekc),]

**To group CHI pellet types properly and graph financial impact of yield**

pellets<-function()**{**

##chi\_yp\_prev<-read.csv("TEST Chicago Daily YP.csv",skip=1)

chi\_yp\_prev<-read.csv("TEST Chicago Daily YP.csv")

**##Average pellet yields for 2017**

##Only 2017 data

chi\_yp17<-filter(chi\_yp\_prev,grepl("2017", chi\_yp\_prev$Yield))

##pel\_typ<-c("S-2","S2","S-3","S3","M-6","M6","M-7","M7","M-8","M8","M-5","ME","L6A","M5","S3B","ST","CRA")

pel\_typ<-c("S","M","L","C","G","H","T","R","B")

pel\_cols<-c(22,27,32,37,42,47,52,57,62)

fila<-vector()

typ<-vector()

colu<-vector()

flr<-data.frame()

m<-0

n<-0

for(j in 1:length(pel\_cols))**{**

for(k in 1:nrow(chi\_yp))**{**

##Comprobar que hay coincidencia

for(i in 1:length(pel\_typ))**{**

if(length(grep(pel\_typ[i],as.character((chi\_yp[k,pel\_cols[j]])),ignore.case=TRUE))!=0)**{**

##guardar el numero de la fila y columna en la que hay coincidencia

##en posiciones sucesivas de respectivos vectores

m<-m+1

colu[m]<-pel\_cols[j]

fila[m]<-k

break

**}**

**}**

**}**

**}**

##build dataframe with every row in fila[m] and column colu[m] plus the next three

for(i in 1:m)**{**

#para cada fila seleccionada…

#sacar fecha

flr[i,1]<-as.Date(chi\_yp17[fila[i],2], "%m/%d/%Y")

d<-substr(as.character(chi\_yp17[fila[i],colu[i]]),1,nchar(as.character(chi\_yp17[fila[i],colu[i]])))

flr[i,2]<-d

for(j in 1:3)**{**

#para cada fila seleccionada…

#sacar columna del ítem

#sacar las tres columnas siguientes (las libras)

c<-parse\_number( chi\_yp17[fila[i],colu[i]+j] )

flr[i,j+2]<-c

##flr[i,j+2]<- chi\_yp17[fila[i],colu[i]+j]

**}**

**}**

colnames(flr)<-c("date","type","lbcut","lbcooked","lbobtained")

##write.csv(flr,"flr.csv")

flr\_tes<-filter(flr,!grepl('tes',type,ignore.case=TRUE))

flr\_tesdat<-filter(flr\_tes,grepl('201',date))

write.csv(flr\_tesdat,"flr\_tesdat.csv")

##remove NA to clean the original table completely

flr\_tesdat1<-select(flr\_tesdat,-lbcut)

flr\_tdc<-flr\_tesdat1[complete.cases(flr\_tesdat1),]

##step to help build yield\_key file

for\_key17<- summarize(group\_by(arrange(flr\_tdc,type),type),sumc=sum(lbcooked,na.rm=TRUE),sumo=sum(lbobtained,na.rm=TRUE),y=(sumo/sumc))

write.csv(for\_key17,"for\_key17.csv")

##Prepare for merging by selecting only the columns that we are interested in

flr\_key<-select(read.csv("yield\_key.csv"),type,fet)

##Merge, no need to write column names "…"

##The key is written manually on the yield\_key.csv file (at least the first time)

flr\_mrg<-merge(flr\_tdc,flr\_key,all.x=TRUE)

flr\_mrgg17<-arrange(group\_by(flr\_mrg,fet,date),fet,date)

write.csv(flr\_mrgg17,"flr\_mrgg17.csv")

alfa<-select(flr\_mrgg17,-type)

beta<-mutate(alfa,weekc=as.character(ymd(date)+(1-wday(ymd(date)))))

write.csv(beta,"beta.csv")

gamma<-read.csv("beta.csv")[,3:6]

delta<-group\_by(gamma,fet,weekc)

flr\_mrgg\_weekc17<-summarize(delta, sumc=sum(lbcooked,na.rm=TRUE),sumo=sum(lbobtained,na.rm=TRUE),yield=(sumo/sumc))

##write.csv(flr\_mrgg\_weekc,"flr\_mrgg\_weekc.csv")

##yield debe ser construido a partir de flr\_mrgg para que muestre las categorias refinadas

yield17<-arrange(summarize(group\_by(arrange(flr\_mrgg17,fet),fet),sumc=sum(lbcooked,na.rm=TRUE),sumo=sum(lbobtained,na.rm=TRUE),y=(sumo/sumc)),desc(sumo))

write.csv(yield17,"yield17.csv")

##si quisiera sacar el top 6 en volumen

##yield\_fil17<-yield17[1:6,]

yield\_fil17<-yield17

##write.csv(yield\_fil17,"yield\_fil17.csv")

flr\_weekc\_fil17<-merge(flr\_mrgg\_weekc17,yield\_fil17,by="fet")

##write.csv(flr\_weekc\_fil17,"flr\_weekc\_fil27.csv")

##compare list of pellets in for\_key and flr\_key (yield\_key.csv), and alert if there are "un-fetenized" pellet types

alert<-for\_key17$type%in%flr\_key$type

ifelse("FALSE"%in%(for\_key17$type%in%flr\_key$type),print(paste("Unmatched pellet type in for\_key17!",for\_key17$type[grep("FALSE",alert)])),print("Ok"))

**##Calculations on pellet yields for 2018**

##Only 2018 data

chi\_yp<-filter(chi\_yp\_prev,grepl("2018", chi\_yp\_prev$Yield))

##il\_yp<-tbl\_df(chi\_yp)

##pel\_typ<-c("S-2","S2","S-3","S3","M-6","M6","M-7","M7","M-8","M8","M-5","ME","L6A","M5","S3B","ST","CRA")

pel\_typ<-c("S","M","L","C","G","H","T","R","B")

pel\_cols<-c(22,27,32,37,42,47,52,57,62)

fila<-vector()

typ<-vector()

colu<-vector()

flr<-data.frame()

m<-0

n<-0

for(j in 1:length(pel\_cols))**{**

for(k in 1:nrow(chi\_yp))**{**

##Comprobar que hay coincidencia

for(i in 1:length(pel\_typ))**{**

if(length(grep(pel\_typ[i],as.character((chi\_yp[k,pel\_cols[j]])),ignore.case=TRUE))!=0)**{**

##guardar el numero de la fila y columna en la que hay coincidencia

##en posiciones sucesivas de respectivos vectores

m<-m+1

colu[m]<-pel\_cols[j]

fila[m]<-k

break

**}**

**}**

**}**

**}**

write.csv(colu,"colu.csv")

write.csv(fila,"fila.csv")

##build dataframe with every row in fila[m] and column colu[m] plus the next three

for(i in 1:m)**{**

#para cada fila seleccionada…

#sacar fecha

flr[i,1]<-as.Date(chi\_yp[fila[i],2], "%m/%d/%Y")

d<-substr(as.character(chi\_yp[fila[i],colu[i]]),1,nchar(as.character(chi\_yp[fila[i],colu[i]])))

flr[i,2]<-d

for(j in 1:3)**{**

#para cada fila seleccionada…

#sacar columna del ítem

#sacar las tres columnas siguientes (las libras)

c<-parse\_number( chi\_yp[fila[i],colu[i]+j] )

flr[i,j+2]<-c

##flr[i,j+2]<- chi\_yp[fila[i],colu[i]+j]

**}**

**}**

colnames(flr)<-c("date","type","lbcut","lbcooked","lbobtained")

write.csv(flr,"flr.csv")

flr\_tes<-filter(flr,!grepl('tes',type,ignore.case=TRUE))

flr\_tesdat<-filter(flr\_tes,grepl('201',date))

write.csv(flr\_tesdat,"flr\_tesdat.csv")

##remove NA to clean the original table completely

flr\_tesdat1<-select(flr\_tesdat,-lbcut)

flr\_tdc<-flr\_tesdat1[complete.cases(flr\_tesdat1),]

write.csv(flr\_tdc,"flr\_tdc.csv")

##step to help build yield\_key file

for\_key<- summarize(group\_by(arrange(flr\_tdc,type),type),sumc=sum(lbcooked,na.rm=TRUE),sumo=sum(lbobtained,na.rm=TRUE),y=(sumo/sumc))

write.csv(for\_key,"for\_key.csv")

##Prepare for merging by selecting only the columns that we are interested in

flr\_key<-select(read.csv("yield\_key.csv"),type,fet)

##Merge, no need to write column names "…"

##The key is written manually on the yield\_key.csv file (at least the first time)

flr\_mrg<-merge(flr\_tdc,flr\_key,all.x=TRUE)

flr\_mrgg<-arrange(group\_by(flr\_mrg,fet,date),fet,date)

write.csv(flr\_mrgg,"flr\_mrgg.csv")

alfa<-select(flr\_mrgg,-type)

beta<-mutate(alfa,weekc=as.character(ymd(date)+(1-wday(ymd(date)))))

write.csv(beta,"beta.csv")

gamma<-read.csv("beta.csv")[,3:6]

delta<-group\_by(gamma,fet,weekc)

flr\_mrgg\_weekc<-summarize(delta, sumc=sum(lbcooked,na.rm=TRUE),sumo=sum(lbobtained,na.rm=TRUE),yield=(sumo/sumc))

write.csv(flr\_mrgg\_weekc,"flr\_mrgg\_weekc.csv")

##yield debe ser construido a partir de flr\_mrgg para que muestre las categorias refinadas

yield<-arrange(summarize(group\_by(arrange(flr\_mrgg,fet),fet),sumc=sum(lbcooked,na.rm=TRUE),sumo=sum(lbobtained,na.rm=TRUE),y=(sumo/sumc)),desc(sumo))

##y is YTD average yield by pellet type while yield is the weekly yield by pellet type

write.csv(yield,"yield.csv")

##si quisiera sacar el top 6 en volumen

##yield\_fil<-yield[1:6,]

yield\_fil<-yield

write.csv(yield\_fil,"yield\_fil.csv")

flr\_weekc\_fil<-merge(flr\_mrgg\_weekc,yield\_fil,by="fet")

write.csv(flr\_weekc\_fil,"flr\_weekc\_fil.csv")

##compare list of pellets in for\_key and flr\_key (yield\_key.csv), and alert if there are "un-fetenized" pellet types

##alert<-merge(for\_key,flr\_key,by.x="type",by.y="type",all.x=TRUE,sort=TRUE,no.dups=TRUE,incomparables=TRUE)

alert<-for\_key$type%in%flr\_key$type

ifelse("FALSE"%in%(for\_key$type%in%flr\_key$type),print(paste("Unmatched pellet type!",for\_key$type[grep("FALSE",alert)])),print("Ok"))

**##Calculate weekly accumulated savings and plot**

##Convert weekc to date for 2018 data

yw18<-mutate(flr\_weekc\_fil,weekc=as.Date(as.character(weekc)))

write.csv(yw18,"yw18.csv")

##calculate savings

##ch\_sav<-numeric()

ch\_sav<-data.frame()

yield17<-transform(yield17,fetc=as.character(fet))

fet17<-character()

tipo17<-character()

ch\_tipo<-character()

ch\_sem<-character()

ch\_cook<-numeric()

ch\_rend<-numeric()

ch\_obt<-numeric()

y17<-numeric()

rend17<-numeric()

k<-0

m<-0

up<-0

down<-0

##Add '17 yield to the '18 yield table

##yield[,5]<-filter(yield17,yield17[,1]==yield[,1])[,4]

##names(yield)[6]<-"y17"

**## calculo de la weekc, esto de abajo es un dataframe**

**##> dim(weeks2018)**

**##[1] 365 2**

**##la weekc está en la columna 2**

days2018<-seq(from=as.Date("2018-01-01"),to=as.Date("2018-12-31"),by =1)

days2018\_d<-transform(as.Date(days2018))

weeks2018<-mutate(days2018\_d,weekc=days2018\_d[,1]+1-wday(days2018\_d[,1]))

weeks18u<-unique(weeks2018$weekc)

write.csv(weeks18u,"weeks18u.csv")

##for(i in 1:nrow(yw18))**{ ##Mal hecho, tiene que ser un loop para cada semana del anho**

for(i in 1:length(weeks18u))**{**

for(j in 1:nrow(yield))**{**

##prepare to extract values from yw18 and yield17

yw\_t<-yw18[grep(yield$fet[j],yw18$fet),]

yw\_tw<-yw\_t[grep(weeks18u[i],yw\_t$weekc),]

write.csv(yw\_tw,"yw\_tw.csv")

yield17\_t<-yield17[grep(yield$fet[j],yield17$fet),]

ch\_sav[j+k,1]<-weeks18u[i] ##weekc

ch\_sav[j+k,2]<-yield$fet[j] ##pel type

ch\_sav[j+k,3]<-ifelse(length(yw\_tw$yield)==0,NA,yw\_tw$yield)

up<- ifelse(length(yw\_tw$yield)==0,0,yw\_tw$sumc.x)

down<- ifelse(length(yw\_tw$yield)==0,0,yw\_tw$sumo.x)

ch\_sav[j+k,4]<- ifelse**(**length(yield17\_t$fet)==0,NA,ifelse**(**i==1,0, ifelse**(is.na(**ch\_sav[j+k-nrow(yield),4]**)**==TRUE,0,ch\_sav[j+k-nrow(yield),4]**))**+0.3\***(**-up+down/yield17\_t$y**))**

ch\_sav[j+k,5]<- ifelse(length(yield17\_t$y)==0,NA,yield17\_t$y)

ch\_sav[j+k,6]<-ifelse**(**i==1,0, ifelse**(is.na(**ch\_sav[j+k-nrow(yield),6]**)**==TRUE,0,ch\_sav[j+k-nrow(yield),6]**))+up**

ch\_sav[j+k,7]<- ifelse**(**i==1,0, ifelse**(is.na(**ch\_sav[j+k-nrow(yield),7]**)**==TRUE,0,ch\_sav[j+k-nrow(yield),7]**))+down**

ch\_sav[j+k,8]<-ifelse(length(yw\_tw$yield)==0,0, ch\_sav[j+k,7]\* as.numeric(ch\_sav[j+k,2]))

**}**

k<-i\*(nrow(yield))

**}**

names(ch\_sav)<-c("weekc","type\_cat","yield","acc\_sav","y17","lbc","lbo","wei\_y18")

##comprobaciones varias

write.csv(ch\_sav,"ch\_sav.csv")

write.csv(yield17,"yield17.csv")

##Add all savings from all pellet types

ch\_totsav\_ytd<-summarize(group\_by(ch\_sav,weekc),sum(acc\_sav,na.rm=TRUE))

names(ch\_totsav\_ytd)<-c("weekc","ch\_acc\_sav")

ch\_pelsav\_ytd<-summarize(group\_by(ch\_sav,type\_cat),sum(lbc,na.rm=TRUE),sum(lbo,na.rm=TRUE),sum(wei\_y18,na.rm=TRUE))

names(ch\_pelsav\_ytd)<-c("type\_cat","totlbcooked","totlbobtained","lbo\_x\_y")

mean\_yield\_18<-ch\_pelsav\_ytd$lbo\_x\_y/ ch\_pelsav\_ytd$totlbobtained

write.csv(ch\_totsav\_ytd," ch\_totsav\_ytd.csv")

ch\_totsav\_ytd<-mutate(ch\_totsav\_ytd,ch\_acc\_sav2= formatC**(**as.numeric**(**ch\_totsav\_ytd$ch\_acc\_sav**)**,format="f",digits=0,big.mark=","**)**)

ch\_sav1<-arrange(ch\_sav,type\_cat,weekc)

ch\_totsav\_ytd1<-select(ch\_totsav\_ytd,weekc,ch\_acc\_sav)

names(ch\_totsav\_ytd1)<-c("weekc","acc\_sav")

ch\_totsav\_ytd1$type\_cat<-"Chicago"

ch\_sav\_s<-rbind(select(ch\_totsav\_ytd1,weekc,type\_cat,acc\_sav),select(ch\_sav1,weekc,type\_cat,acc\_sav))

write.csv(ch\_sav\_s,"ch\_sav\_s.csv")

ch\_sav\_s<-read.csv("ch\_sav\_s.csv")

View(ch\_sav\_s)

##dev.new()

##plot.new()

dev.new()

plot.new()

ifelse(min(ch\_sav$acc\_sav,ch\_totsav\_ytd$ch\_acc\_sav,na.rm=TRUE)>0,y1<-0,y1<- min(ch\_sav$acc\_sav,ch\_totsav\_ytd$ch\_acc\_sav,na.rm=TRUE))

ifelse(max(ch\_sav$acc\_sav,ch\_totsav\_ytd$ch\_acc\_sav,na.rm=TRUE)<0,y2<-0,y2<- max(ch\_sav$acc\_sav,ch\_totsav\_ytd$ch\_acc\_sav,na.rm=TRUE))

pts <-seq(y1,y2,by=(y2-y1)/4)

ptsv<-seq(0,length(ch\_sav$weekc),by=1)

plot(as.Date(ch\_sav$weekc),as.numeric(ch\_sav$acc\_sav),type="n",col="brown",lwd=2,xlab="2018",ylab="Accumulated savings (usd)",main="Chicago pellet yield savings 2018 vs 2017",ylim=c(y1,y2),yaxt="n",bty="n")

points(as.Date(filter(ch\_sav,type\_cat=="FB")$weekc),as.numeric(filter(ch\_sav,type\_cat=="FB")$acc\_sav),col="gray",type="l",lwd=2, bty="n")

points(as.Date(filter(ch\_sav,type\_cat=="M5B")$weekc),as.numeric(filter(ch\_sav,type\_cat=="M5B")$acc\_sav),col="green",type="l",lwd=2, bty="n")

points(as.Date(filter(ch\_sav,type\_cat=="CR")$weekc),as.numeric(filter(ch\_sav,type\_cat=="CR")$acc\_sav),col="red",type="l",lwd=2, bty="n")

points(as.Date(filter(ch\_sav,type\_cat=="ST")$weekc),as.numeric(filter(ch\_sav,type\_cat=="ST")$acc\_sav),col="brown",type="l",lwd=2, bty="n")

points(as.Date(filter(ch\_sav,type\_cat=="Sma")$weekc),as.numeric(filter(ch\_sav,type\_cat=="Sma")$acc\_sav),col="blue",type="l",lwd=2, bty="n")

points(as.Date(ch\_totsav\_ytd$weekc),as.numeric(ch\_totsav\_ytd$ch\_acc\_sav),col="purple",type="l",lwd=3, bty="n")

legend("bottomleft",lty=1:1,col=c("gray","green","red","brown","blue","purple"),legend=c("FB","M5B","CR","ST","Sma","Chicago"),ncol=2,lwd=3)

abline(h=pts,v=ptsv,col=" lightgray", lwd=1,lty="dotted")

abline(h=0,col=" lightgray", lwd=2,lty="dotted")

abline(h= as.numeric(ch\_totsav\_ytd[length(ch\_totsav\_ytd$ch\_acc\_sav),2]),col=" lightgray", lwd=2,lty="dotted")

abline(v=as.Date(ymd(today())+(1-wday(ymd(today())))),col=" lightgray", lwd=2,lty="dotted")

axis**(**2, at = c**(**as.numeric**(**ch\_totsav\_ytd[length(ch\_totsav\_ytd$ch\_acc\_sav),2]**)**,0,pts**)**, labels = c(formatC**(**as.numeric**(**ch\_totsav\_ytd[length(ch\_totsav\_ytd$ch\_acc\_sav),2]/1000**)**,format="f",digits=0,big.mark=","**)**,0,paste**(**formatC**(**pts/1000,format="f",digits=0,big.mark=","**)**, "k", sep = ""**)**),las=1**)**

axis**(**1, at = as.Date(ymd(today())+(1-wday(ymd(today())))), labels = "today",las=2**)**

pause = function()

{

    if (interactive())

    {

        invisible(readline(prompt = " Please copy the graph, click on this window and press <Enter> to continue..."))

    }

    else

    {

        cat("Please copy the graph, click on this window and press <Enter> to continue...")

        invisible(readLines(file("stdin"), 1))

    }

}

pause()

dev.new()

plot.new()

**g<-ggplot(data=filter(ch\_sav\_s,type\_cat!="San"), aes(x=as.Date(weekc), y= acc\_sav,color=type\_cat,shape=type\_cat))**

**##g+geom\_point()**

**g+geom\_point()+geom\_path()+theme\_bw()+labs(x="2018",y="USD",title="Chicago pellet yield savings (usd) 2018 vs 2017")+** theme(plot.title = element\_text(hjust = 0.5))**+**geom\_vline(xintercept= as.Date(ymd(today())+(1-wday(ymd(today())))),linetype="dashed")+ geom\_hline(yintercept= as.numeric(subset(ch\_sav\_s,type\_cat=="M5B"&weekc==ch\_sav\_s$weekc[length(unique(ch\_sav\_s$weekc))],select=acc\_sav)[1,1]),linetype="dashed")+geom\_hline(yintercept=as.numeric(subset(ch\_sav\_s,type\_cat=="Chicago"&weekc==ch\_sav\_s$weekc[length(unique(ch\_sav\_s$weekc))],select=acc\_sav)[1,1]),color="blue",size=1.25)+scale\_y\_continuous**(**labels = scales::comma,breaks=c**(**min(ch\_sav\_s $acc\_sav, na.rm=TRUE), max(ch\_sav\_s $acc\_sav,na.rm=TRUE),0,as.numeric(subset**(**ch\_sav\_s,type\_cat=="Chicago"&weekc==ch\_sav\_s$weekc[length(unique(ch\_sav\_s$weekc))],select=acc\_sav**)**[1,1]), as.numeric(subset(ch\_sav\_s,type\_cat=="ST"&weekc==ch\_sav\_s$weekc[length(unique(ch\_sav\_s$weekc))],select=acc\_sav)[1,1]), as.numeric(subset(ch\_sav\_s,type\_cat=="FB"&weekc==ch\_sav\_s$weekc[length(unique(ch\_sav\_s$weekc))],select=acc\_sav)[1,1]), as.numeric(subset(ch\_sav\_s,type\_cat=="CR"&weekc==ch\_sav\_s$weekc[length(unique(ch\_sav\_s$weekc))],select=acc\_sav)[1,1]), as.numeric(subset(ch\_sav\_s,type\_cat=="Sma"&weekc==ch\_sav\_s$weekc[length(unique(ch\_sav\_s$weekc))],select=acc\_sav)[1,1])**)**)+ geom\_hline(yintercept=0,linetype="dashed")+geom\_hline(yintercept= as.numeric(subset(ch\_sav\_s,type\_cat=="ST"&weekc==ch\_sav\_s$weekc[length(unique(ch\_sav\_s$weekc))],select=acc\_sav)[1,1]),linetype="dashed")+ geom\_hline(yintercept= as.numeric(subset(ch\_sav\_s,type\_cat=="FB"&weekc==ch\_sav\_s$weekc[length(unique(ch\_sav\_s$weekc))],select=acc\_sav)[1,1]),linetype="dashed")+ geom\_hline(yintercept= as.numeric(subset(ch\_sav\_s,type\_cat=="CR"&weekc==ch\_sav\_s$weekc[length(unique(ch\_sav\_s$weekc))],select=acc\_sav)[1,1]),linetype="dashed")+ geom\_hline(yintercept= as.numeric(subset(ch\_sav\_s,type\_cat=="Sma"&weekc==ch\_sav\_s$weekc[length(unique(ch\_sav\_s$weekc))],select=acc\_sav)[1,1]),linetype="dashed")

**##+theme(axis.text.x = element\_text(face="bold", color="#993333", size=9, angle=45),axis.text.y = element\_text(face="bold", color="#993333", size=9, angle=45))+scale\_y\_continuous(labels = scientific)**

**}**

**End of function pellets()**

# [**How do I change the formatting of numbers on an axis with ggplot?**](https://stackoverflow.com/questions/11610377/how-do-i-change-the-formatting-of-numbers-on-an-axis-with-ggplot)

Another option is to format your axis tick labels with commas is by using the package scales, and add

scale\_y\_continuous(name="Fluorescent intensity/arbitrary units", labels = comma)

to your ggplot statement.

If you don't want to load the package, use:

scale\_y\_continuous(name="Fluorescent intensity/arbitrary units", labels = scales::comma)

Ignore column names with rbind

rbind(df,setNames(df[,2:1],names(df)))

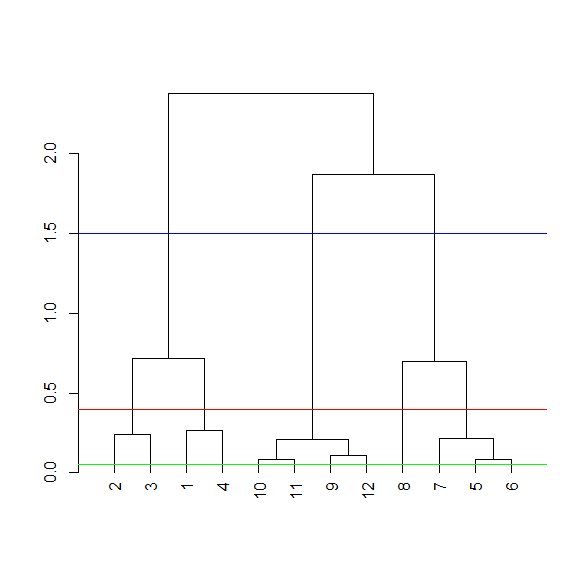
Hierarchical Clustering

hc<-hclust(distxy)

plot(hc)

plot(as.dendrogram(hc)) ##(Notice that the vertical heights of the lines and labeling of the scale on the left edge give some indication of distance).

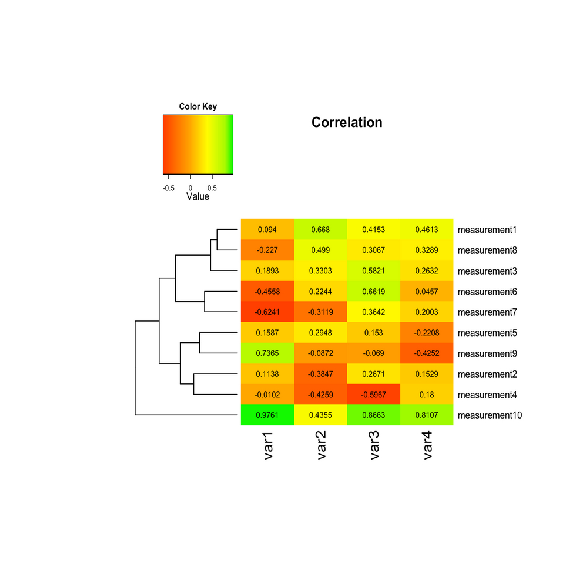
abline(h=1.5,col="blue") ## We call this a "cut" of our dendrogram



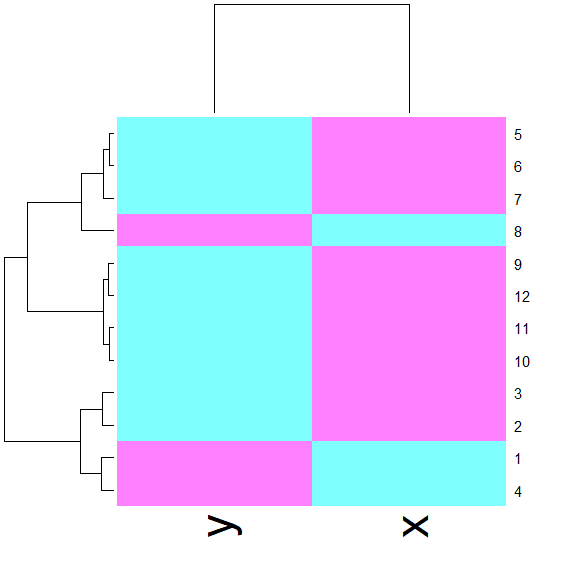
how distances between clusters of points are measured, we'll just mention two. The first is called complete linkage and it says that if you're trying to measure a distance between two clusters, take the greatest distance between the pairs of points in those two clusters. Second, average linkage. First you compute an "average" point in each cluster (think of it as the cluster's center of gravity). You do this by computing the mean (average) x and y coordinates of the points in the cluster. In our simple set of data, the average and complete linkages aren't that different, but in more complicated datasets the type of linkage you use could affect how your data clusters.

It is a good idea to experiment with different methods of linkage to see the varying ways your data groups.

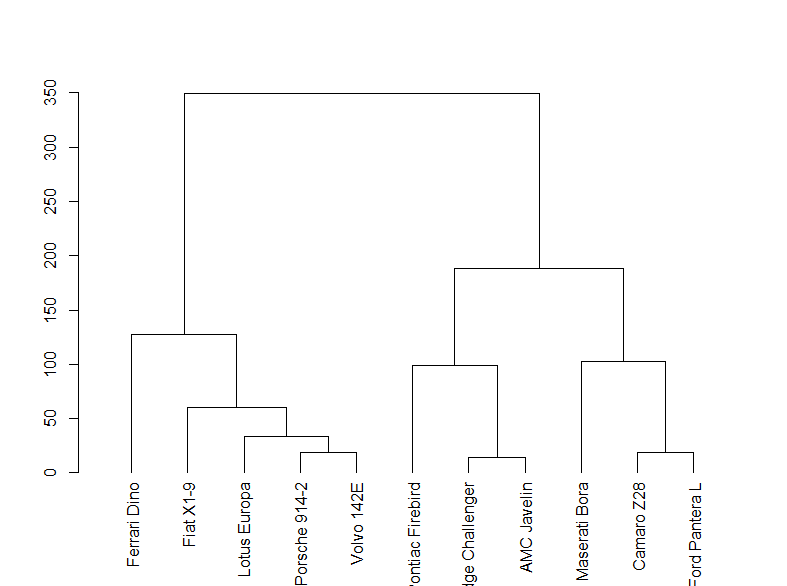
Heatmaps: a graphical representation of data where the individual values contained in a matrix are represented as colors. ... Heat maps originated in 2D displays of the values in a data matrix. Larger values were represented by small dark gray or black squares (pixels) and smaller values by lighter squares."



heatmap(dataMatrix,col=cm.colors(25))



|  |  |
| --- | --- |
| heatmap(mt)  mt  mpg cyl disp hp drat wt  Dodge Challenger 15.5 8 318.0 150 2.76 3.520  AMC Javelin 15.2 8 304.0 150 3.15 3.435  Camaro Z28 13.3 8 350.0 245 3.73 3.840  Pontiac Firebird 19.2 8 400.0 175 3.08 3.845  Fiat X1-9 27.3 4 79.0 66 4.08 1.935  Porsche 914-2 26.0 4 120.3 91 4.43 2.140  Lotus Europa 30.4 4 95.1 113 3.77 1.513  Ford Pantera L 15.8 8 351.0 264 4.22 3.170  Ferrari Dino 19.7 6 145.0 175 3.62 2.770  Maserati Bora 15.0 8 301.0 335 3.54 3.570  Volvo 142E 21.4 4 121.0 109 4.11 2.780 |  |



**##To re-define pellet expiration dates**

**##Only input is file peltypexp.csv**

pelexp<-read.csv("peltypexp.csv")

names(pelexp)<-c("no","desc","w","exp","item","sl")

pelexp<-pelexp

type<-as.character()

sale<-as.character()

expy<-as.character()

ship<-as.character()

for(i in 1:nrow(pelexp))**{**

type[i]<-ifelse**(**length(grep("REG",pelexp$desc[i],ignore.case=TRUE))!=0,"REG", ifelse**(**length(grep("RTE ",pelexp$desc[i],ignore.case=TRUE))!=0,"RTE",ifelse**(**length(grep("FIN",pelexp$desc[i],ignore.case=TRUE))!=0,"CR",

ifelse**(**length(grep("CRA",pelexp$desc[i],ignore.case=TRUE))!=0,"CR", ifelse**(**length(grep("CURL",pelexp$desc[i],ignore.case=TRUE))!=0,"CR", ifelse**(**length(grep("TAIL",pelexp$desc[i],ignore.case=TRUE))!=0,"CR", ifelse**(**length(grep("fee",pelexp$desc[i],ignore.case=TRUE))!=0,"PI", ifelse**(**length(grep("ear",pelexp$desc[i],ignore.case=TRUE))!=0,"PI",

ifelse**(**length(grep("pick",pelexp$desc[i],ignore.case=TRUE))!=0,"PI", ifelse**(**length(grep("SM",pelexp$desc[i],ignore.case=TRUE))!=0,"REG", ifelse**(**length(grep("STR",pelexp$desc[i],ignore.case=TRUE))!=0,"CR", ifelse**(**length(grep("PLAT",pelexp$desc[i],ignore.case=TRUE))!=0,"CR","REG"))))))))))))

sale[i]<- ifelse**(**length(grep("EXP",pelexp$desc[i],ignore.case=TRUE))!=0,"EXP", ifelse**(**length(grep("20+",pelexp$desc[i],ignore.case=TRUE))!=0,"DOM", ifelse**(**length(grep("US",pelexp$desc[i],ignore.case=TRUE))!=0,"DOM", ifelse**(**length(grep("GUADA",pelexp$desc[i],ignore.case=TRUE))!=0,"EXP", ifelse**(**length(grep("COLU",pelexp$desc[i],ignore.case=TRUE))!=0,"EXP", ifelse**(**length(grep("GUATE",pelexp$desc[i],ignore.case=TRUE))!=0,"EXP", ifelse**(**length(grep("DINA",pelexp$desc[i],ignore.case=TRUE))!=0,"EXP",

ifelse**(**length(grep("MX",pelexp$desc[i],ignore.case=TRUE))!=0,"EXP","DOM"))))))))

**}**

pelexp$type<-type

pelexp$sale<-sale

for(i in 1:nrow(pelexp))**{**

expy[i]<- ifelse**(**pelexp$type[i]=="CR",ifelse**(**pelexp$sale[i]=="DOM","12M","6M"**)**, ifelse**(**pelexp$type[i]=="REG", ifelse**(**pelexp$sale[i]=="DOM","18M","12M"**)**, ifelse**(**pelexp$type[i]=="PI","100D", ifelse**(**pelexp$type[i]=="RTE","3M",NA))))

ship[i]<- ifelse**(**pelexp$type[i]=="CR",ifelse**(**pelexp$sale[i]=="DOM","9M","4M"**)**, ifelse**(**pelexp$type[i]=="REG", ifelse**(**pelexp$sale[i]=="DOM","12M","6M"**)**,NA))

**}**

pelexp$expy<-expy

pelexp$ship<-ship

write.csv(pelexp,"pelexp.csv")

We can use the R function tapply which applies "a function over a ragged array". This means that every element of the array is assigned a factor and the function is applied to subsets of the array (identified by the factor vector). This allows us to take advantage of the factor vector newClust we calculated. Call tapply now with 3 arguments, x (the data), newClust (the factor array), and mean (the function to apply).

> tapply(x,newClust,mean)

1 2 3

1.210767 1.010320 2.498011

Now that you've gone through an example step by step, you'll be relieved to hear that R provides a command to do all this work for you. Unsurprisingly it's called kmeans and, although it has several parameters, we'll just mention four. These are x, (the numeric matrix of data), centers, iter.max, and nstart. The second of these (centers) can be either a number of clusters or a set of initial centroids. The third, iter.max, specifies the maximum number of iterations to go through, and nstart is the number of random starts you want to try if you specify centers as a number.

Call kmeans now with 2 arguments, dataFrame (which holds the x and y coordinates of our 12 points) and centers set equal to 3.

> kmeans(dataFrame,centers=3)

K-means clustering with 3 clusters of sizes 2, 8, 2

The program returns the information that the data clustered into 3 clusters each of size 4. It also returns the coordinates of the 3 cluster means, a vector named cluster indicating how the 12 points were partitioned into the clusters, and the sum of squares within each cluster. It also shows all the available components returned by the function. We've stored off this data for you in a kmeans object called kmObj. Look at kmObj$iter to see how many iterations the algorithm went through.

Two iterations as we did before. We just want to emphasize how you can access the information available to you. Let's plot the data points color coded according to their cluster. This was stored in kmObj$cluster. Run plot with 5 arguments. The data, x and y, are the first two; the third, col is set equal to kmObj$cluster, and the last two are pch and cex. The first of these should be set to 19 and the last to 2.

> plot(x,y,col=kmObj$cluster,pch=19,cex=2)

Now add the centroids which are stored in kmObj$centers. Use the points function with 5 arguments. The first two are kmObj$centers and col=c("black","red","green"). The last three, pch, cex, and lwd, should all| equal 3.

> points(kmObj$centers,col=c("black","red","green"),pch=3,cex=3,lwd=3)

Now for some fun! We want to show you how the output of the kmeans function is affected by its random start (when you just ask for a number of clusters). With random starts you might want to run the function several times to get an idea of the relationships between your observations. We'll call kmeans with the same data points (stored in dataFrame), but ask for 6 clusters instead of 3.

We'll plot our data points several times and each time we'll just change the argument col which will show us how the R function kmeans is clustering them. So, call plot now with 5 arguments. The first 2 are x and y. The third is col set equal to the call kmeans(dataFrame,6)$cluster. The last two (pch and cex) are set to 19 and 2 respectively.

> plot(x,y,col=kmeans(dataFrame,6)$cluster,pch=19,cex=2)

swirl

heatmap(dataMatrix)

addPatt.R

set.seed(678910)

for(i in 1:40){

# flip a coin

coinFlip <- rbinom(1,size=1,prob=0.5)

# if coin is heads add a common pattern to that row

if(coinFlip){

dataMatrix[i,] <- dataMatrix[i,] + rep(c(0,3),each=5)

}

}

| So whether or not a row gets modified by a pattern is determined by a coin

| flip. Will the added pattern affect every column in the affected row?

1: Yes

2: No

Selection: 1

| Not quite! Try again.

| The expression rep(c(0,3),each=5) creates the 10-long vector

| (0,0,0,0,0,3,3,3,3,3) which is added to the rows chosen by the coin flip.

1: Yes

2: No

| Now to execute this code, run the R command source with 2 arguments. The

| first is the filename (in quotes), "addPatt.R", and the second is the

| argument local set equal to TRUE.

> source("addPatt.R",local=TRUE)

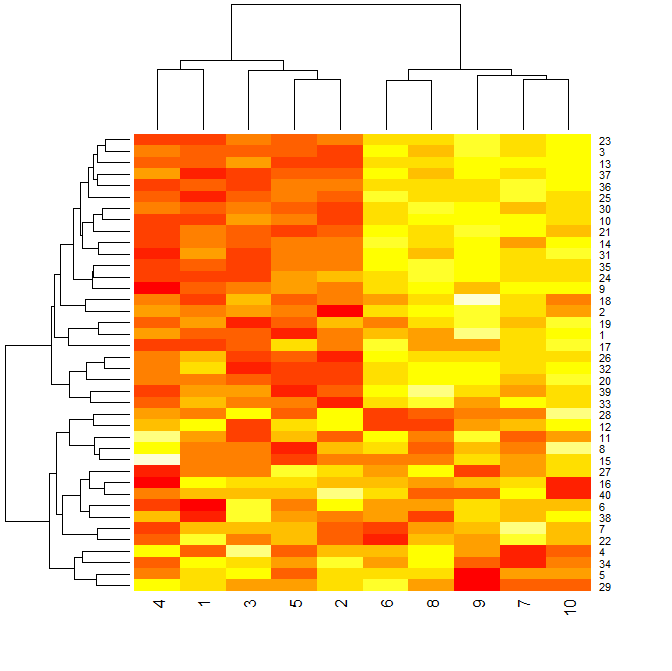
Again we see the pattern in the columns of the matrix. As shown in the

| dendrogram at the top of the display, these split into 2 clusters, the lower

| numbered columns (1 through 5) and the higher numbered ones (6 through 10).

| Recall from the code in addPatt.R that for rows selected by the coinflip the

| last 5 columns had 3 added to them. The rows still look random.



Now consider this picture. On the left is an image similar to the heatmap of

| dataMatix you just plotted. It is an image plot of the output of hclust(), a

| hierarchical clustering function applied to dataMatrix. Yellow indicates

| "hotter" or higher values than red. This is consistent with the pattern we

| applied to the data (increasing the values for some of the rightmost

| columns).

| The middle display shows the mean of each of the 40 rows (along the x-axis).

| The rows are shown in the same order as the rows of the heat matrix on the

| left. The rightmost display shows the mean of each of the 10 columns. Here

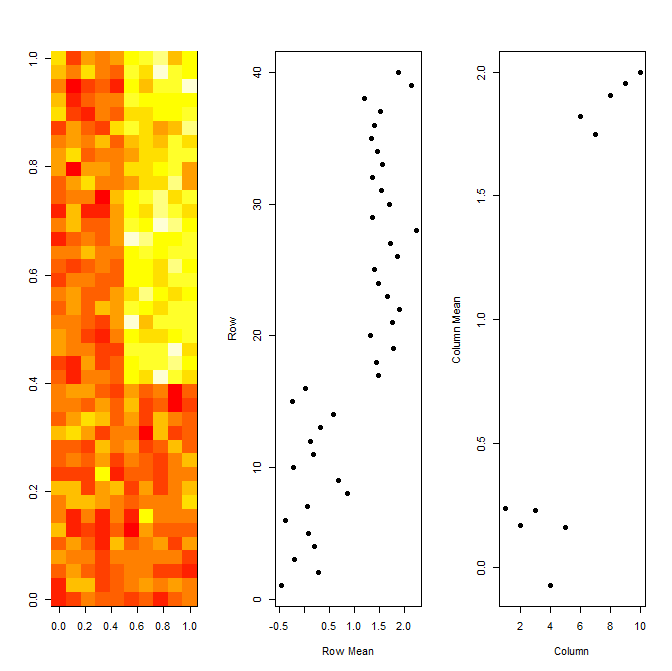
| the column numbers are along the x-axis and their means along the y.

| We see immediately the connection between the yellow (hotter) portion of the

| cluster image and the higher row means, both in the upper right portion of

| the displays. Similarly, the higher valued column means are in the right half

| of that display and lower colummn means are in the left half.



| Now we'll talk a little theory. Suppose you have 1000's of multivariate

| variables X\_1, ... ,X\_n. By multivariate we mean that each X\_i contains many

| components, i.e., X\_i = (X\_{i1}, ... , X\_{im}. However, these variables

| (observations) and their components might be correlated to one another.

| As data scientists, we'd like to find a smaller set of multivariate variables

| that are uncorrelated AND explain as much variance (or variability) of the

| data as possible. This is a statistical approach.

| In other words, we'd like to find the best matrix created with fewer

| variables (that is, a lower rank matrix) that explains the original data.

| This is related to data compression.

| Two related solutions to these problems are PCA which stands for Principal

| Component Analysis and SVD, Singular Value Decomposition. This latter simply

| means that we express a matrix X of observations (rows) and variables

| (columns) as the product of 3 other matrices, i.e., X=UDV^t. This last term

| (V^t) represents the transpose of the matrix V.

Here U and V each have orthogonal (uncorrelated) columns. U's columns are the

| left singular vectors of X and V's columns are the right singular vectors of

| X. D is a diagonal matrix, by which we mean that all of its entries not on

| the diagonal are 0. The diagonal entries of D are the singular values of X.

| To illustrate this idea we created a simple example matrix called mat. Look

| at it now.

> mat

[,1] [,2] [,3]

[1,] 1 2 3

[2,] 2 5 7

So mat is a 2 by 3 matrix. Lucky for us R provides a function to perform singular value decomposition. It's called, unsurprisingly, svd. Call it now with a single argument, mat.

> svd(mat)

$`d`

[1] 9.5899624 0.1806108

$u

[,1] [,2]

[1,] -0.3897782 -0.9209087

[2,] -0.9209087 0.3897782

$v

[,1] [,2]

[1,] -0.2327012 -0.7826345

[2,] -0.5614308 0.5928424

[3,] -0.7941320 -0.1897921

We see that the function returns 3 components, d which holds 2 diagonal elements, u, a 2 by 2 matrix, and v, a 3 by 2 matrix. We stored the diagonal entries in a diagonal matrix for you, diag, and we also stored u and v in the variables matu and matv respectively. Multiply matu by diag by t(matv) to see what you get. (This last expression represents the transpose of matv in R). Recall that in R matrix multiplication requires you to use the operator %\*%.

> matu %\*% diag %\*% t(matv)

[,1] [,2] [,3]

[1,] 1 2 3

[2,] 2 5 7

Now we'll talk a little about PCA, Principal Component Analysis, "a simple, non-parametric method for extracting relevant information from confusing data sets." We're quoting here from a very nice concise paper on this subject which can be found at http://arxiv.org/pdf/1404.1100.pdf. The paper by Jonathon Shlens of Google Research is called, A Tutorial on Principal Component Analysis. Basically, PCA is a method to reduce a high-dimensional data set to its essential elements (not lose information) and explain the variability in the data. We won't go into the mathematical details here, (R has a function to perform PCA), but you should know that SVD and PCA are closely related.

We'll demonstrate this now. First we have to scale mat, our simple example data matrix. This means that we subtract the column mean from every element and divide the result by the column standard deviation. Of course R has a command, scale, that does this for you. Run svd on scale of mat.

> svd(scale(mat))

$`d`

[1] 1.732051 0.000000

$u

[,1] [,2]

[1,] -0.7071068 0.7071068

[2,] 0.7071068 0.7071068

$v

[,1] [,2]

[1,] 0.5773503 -0.5773503

[2,] 0.5773503 0.7886751

[3,] 0.5773503 -0.2113249

Now run the R program prcomp on scale(mat). This will give you the principal components of mat. See if they look familiar.

> prcomp(scale(mat))

Standard deviations (1, .., p=2):

[1] 1.732051 0.000000

Rotation (n x k) = (3 x 2):

PC1 PC2

[1,] 0.5773503 -0.5773503

[2,] 0.5773503 0.7886751

[3,] 0.5773503 -0.2113249

Notice that the principal components of the scaled matrix, shown in the Rotation component of the prcomp output, ARE the columns of V, the right singular values. Thus, PCA of a scaled matrix yields the V matrix (right singular vectors) of the same scaled matrix.

To prove we're not making this up, we've run svd on dataMatrix and stored the result in the object svd1. This has 3 components, d, u and v. look at the first column of V now. It can be viewed by using the svd1$v[,1] notation.

> svd1$v[,1]

Why were the first columns of both the U and V matrices so special? Well as it happens, the D matrix of the SVD explains this phenomenon. It is an aspect of SVD called variance explained. Recall that D is the diagonal matrix sandwiched in between U and V^t in the SVD representation of the data matrix. The diagonal entries of D are like weights for the U and V columns accounting for the variation in the data. They're given in decreasing order from highest to lowest. Look at these diagonal entries now. Recall that they're stored in svd1$d.

> svd1$d

[1] 12.458121 7.779798 6.732595 6.301878 5.860013 4.501826 3.921267

[8] 2.973909 2.401470 2.152848

Here's a display of these values (on the left). The first one (12.46) is significantly bigger than the others. Since we don't have any units specified, to the right we've plotted the proportion of the variance each entry represents. We see that the first entry accounts for about 40% of the variance in the data. This explains why the first columns of the U and V matrices respectively showed the distinctive patterns in the row and column means so clearly.

According to the plot, what percentage of the total variation does the first diagonal element account for?

1: 90%

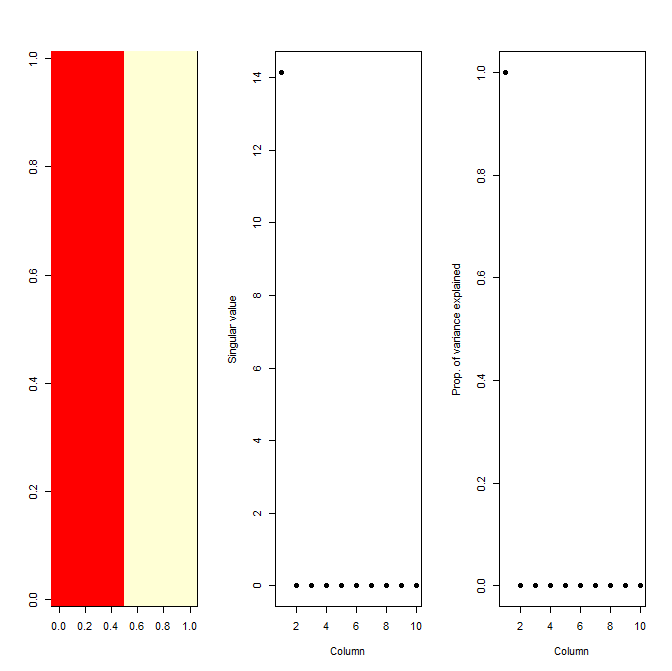
2: 100%

3: 0%

4: 50%

Selection: 2

So what does this mean? Basically that the data is one-dimensional. Only 1 piece of information, namely which column an entry is in, determines its value.



To see this more closely, look at the first 2 columns of the v component. We stored the SVD output in the svd object svd2.

> svd2$v[,1:2]

[,1] [,2]

[1,] 0.06154540 0.142468636

[2,] 0.26433096 0.504510087

[3,] 0.04987554 0.316470664

[4,] 0.27693897 0.524499356

[5,] 0.14275820 -0.282921362

[6,] 0.43252652 -0.002280468

[7,] 0.37724057 -0.354403893

[8,] 0.43280767 0.039226153

[9,] 0.34912246 -0.376485206

[10,] 0.43379723 -0.031422705

So the first element which showed the difference between the left and right halves of the matrix accounts for roughly 50% of the variation in the matrix, and the second element which picked up the alternating pattern accounts for 18% of the variance. The remaining elements account for smaller percentages of the variation. This indicates that the first pattern is much stronger than the second. Also the two patterns confound each other so they're harder to separate and see clearly. This is what often happens with real data.

Now you're probably convinced that SVD and PCA are pretty cool and useful as

| tools for analysis, but one problem with them that you should be aware of, is

| that they cannot deal with MISSING data. Neither of them will work if any

| data in the matrix is missing. (You'll get error messages from R in red if

| you try.) Missing data is not unusual, so luckily we have ways to work around

| this problem. One we'll just mention is called imputing the data.

This uses the k nearest neighbors to calculate a values to use in place of

| the missing data. You may want to specify an integer k which indicates how

| many neighbors you want to average to create this replacement value. The

| bioconductor package (http://bioconductor.org) has an impute package which

| you can use to fill in missing data. One specific function in it is

| impute.knn.

We'll move on now to a final example of the power of singular value

| decomposition and principal component analysis and how they work as a data

| compression technique.

Suppose we create the product of pieces of these, say the first columns of U

| and V and the first element of D. The first column of U can be interpreted as

| a 32 by 1 matrix (recall that faceData was a 32 by 32 matrix), so we can

| multiply it by the first element of D, a 1 by 1 matrix, and get a 32 by 1

| matrix result. We can multiply that by the transpose of the first column of

| V, which is the first principal component. (We have to use the transpose of

| V's column to make it a 1 by 32 matrix in order to do the matrix

| multiplication properly.)

Alas, that is how we do it in theory, but in R using only one element of d

| means it's a constant. So we have to do the matrix multiplication with the

| %\*% operator and the multiplication by the constant (svd1$d[1]) with the

| regular multiplication operator \*.

Try this now and put the result in the variable a1. Recall that svd1$u,

| svd1$d, and svd1$v contain all the information you need. NOTE that because of

| the peculiarities of R's casting, if you do the scalar multiplication with

| the \* operator first (before the matrix multiplication with the %\*% operator)

| you MUST enclose the 2 arguments (svd1$u[,1] and svd1$d[1]) in parentheses.

a1 <- svd1$u[,1] %\*% t(svd1$v[,1]) \* svd1$d[1]

Create the matrix a2 as the product of the first 2 columns of svd1$u, a

| diagonal matrix using the first 2 elements of svd1$d, and the transpose of

| the first 2 columns of svd1$v. Since all of your multiplicands are matrices

| you have to use only the operator %\*% AND you DON'T need parentheses. Also,

| you must use the R function diag with svd1$d[1:2] as its sole argument to

| create the proper diagonal matrix. Remember, matrix multiplication is NOT

| commutative so you have to put the multiplicands in the correct order. Please

| use the 1:2 notation and not the c(m:n), i.e., the concatenate function, when

| specifying the columns.

> a2 <- svd1$u[,1:2] %\*% diag(svd1$d[1:2]) %\*% t(svd1$v[,1:2])

> myImage(svd1$u[,1:5] %\*% diag(svd1$d[1:5]) %\*% t(svd1$v[,1:5]))

Singular value decomposition… principal component analysis…

**We'll close now with a few comments. First, when reducing dimensions you have**

**| to pay attention to the scales on which different variables are measured and**

**| make sure that all your data is in consistent units. In other words, scales**

**| of your data matter. Second, principal components and singular values may mix**

**| real patterns, as we saw in our simple 2-pattern example, so finding and**

**| separating out the real patterns require some detective work. Let's do a**

**| quick review now.**

A matrix X has the singular value decomposition UDV^t. The principal components of X are ?

1: the rows of V

2: the rows of U

3: the columns of V

4: the columns of U

Selection: 3

A matrix X has the singular value decomposition UDV^t. The singular values of X are found where?

1: the columns of D

2: the diagonal elements of D

3: the columns of U

4: the columns of V

Selection: 2

D gives the singular values of a matrix in decreasing order of weight.

In this lesson we'll apply some of the analytic techniques we learned in this course to data from the University of California, Irvine. Specifically, the data we'll use is from UCI's Center for Machine Learning and Intelligent Systems. You can find out more about the data at http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones. As this address indicates, the data involves smartphones and recognizing human activity. Cool, right?

Our goal is to show you how to use exploratory data analysis to point you in fruitful directions of research, that is, towards answerable questions. Exploratory data analysis is a "rough cut" or filter which helps you to find the most beneficial areas of questioning so you can set your priorities accordingly.

So we're looking at training data from a machine learning repository. We can infer that this data is supposed to train machines to recognize activity collected from the accelerometers and gyroscopes built into the smartphones that the subjects had strapped to their waists.

names(ssd[,562:563])

[1] "subject" "activity"

Because it's training data, each row is labeled with the correct activity (from the 6 possible) and associated with the column measurements (from the accelerometer and gyroscope). We're interested in questions such as, "Is the correlation between the measurements and activities good enough to train a machine?" so that "Given a set of 561 measurements, would a trained machine be able to determine which of the 6 activities the person was doing?"

First, let's massage the data a little so it's easier to work with. We've already run the R command transform on the data so that activities are factors. This will let us color code them when we generate plots.

Let's look at only the first subject (numbered 1). Create the variable sub1 by assigning to it the output of the R command subset with ssd as the first argument and the boolean, subject equal to 1, as the second.

sub1<-subset(ssd,subject==1)

names(sub1[,1:12])

Make sure your cursor is back in the console window before you hit any more buttons.

> myedit("showXY.R")

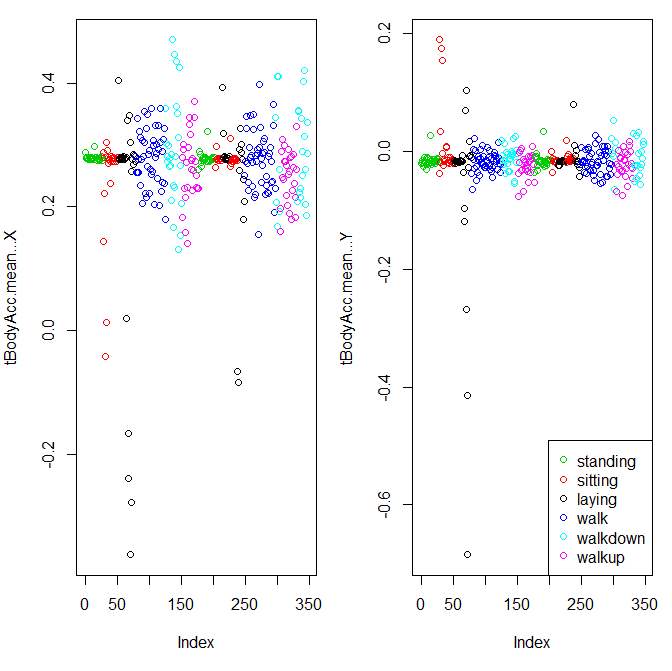
par(mfrow=c(1, 2), mar = c(5, 4, 1, 1))

plot(sub1[, 1], col = sub1$activity, ylab = names(sub1)[1])

plot(sub1[, 2], col = sub1$activity, ylab = names(sub1)[2])

legend("bottomright",legend=unique(sub1$activity),col=unique(sub1$activity), pch = 1)

par(mfrow=c(1,1))



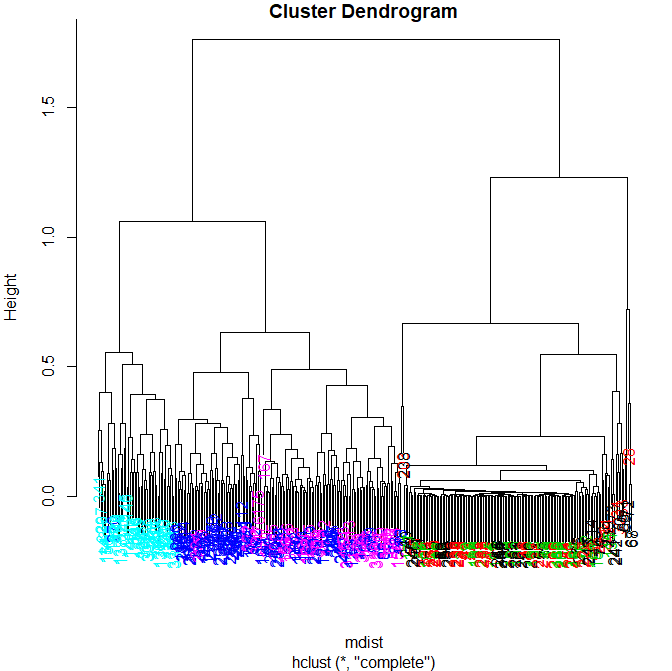
Create a distance matrix, mdist, of the first 3 columns of sub1, by using the R command dist. Use the x[,1:3] notation to specify the columns.

> mdist<-dist(sub1[,1:3])

hclustering<-hclust(mdist) ##Euclidean distance as its default metric

Now call the pretty plotting function (which we've already sourced) myplclust with 2 arguments. The first is hclustering, and the second is the argument lab.col set equal to unclass(sub1$activity).

> myplclust(hclustering,lab.col=unclass(sub1$activity))



svd1<-scale(sub1[,-c(562,563])

svd1<-svd(scale(sub1[,-c(562,563)]))

Here we're looking at the 2 left singular vectors of svd1 (the first 2

| columns of svd1$u). Each entry of the columns belongs to a particular row

| with one of the 6 activities assigned to it. We see the activities

| distinguished by color. Moving from left to right, the first section of rows

| are green (standing), the second red (sitting), the third black (laying),

| etc. The first column of u shows separation of the nonmoving (black, red,

| and green) from the walking activities. The second column is harder to

| interpret. However, the magenta cluster, which represents walking up, seems

| separate from the others.

| We'll try to figure out why that is. To do that we'll have to find which of

| the 500+ measurements (represented by the columns of sub1) contributes to the

| variation of that component. Since we're interested in sub1 columns, we'll

| look at the RIGHT singular vectors (the columns of svd1$v), and in

| particular, the second one since the separation of the magenta cluster stood

| out in the second column of svd1$u.

Here's a plot of the second column of svd1$v. We used transparency in our

| plotting but nothing clearly stands out here. Let's use clustering to find

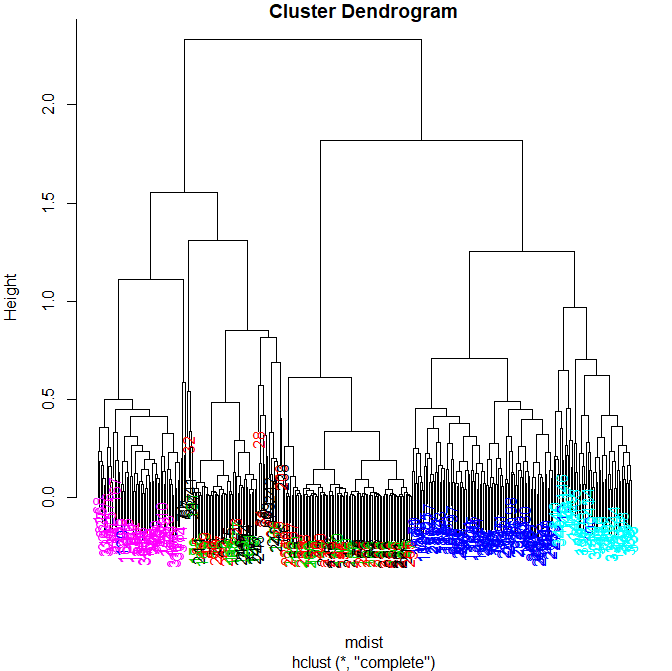
| the feature (out of the 500+) which contributes the most to the variation of

| this second column of svd1$v.

maxCon<-which.max(svd1$v[,2])

mdist <- dist(sub1[,c(10:12,maxCon)])

hclustering<-hclust(mdist)



Run the R command names with the argument sub1[maxCon] to see what measurement is associated with this maximum contributor.

> names(sub1[maxCon])

[1] "fBodyAcc.meanFreq...Z"

table(kClust$cluster,sub1$activity)

laying sitting standing walk walkdown walkup

1 24 33 46 0 0 0

2 0 0 0 0 49 0

3 10 2 0 0 0 0

4 0 0 0 95 0 0

5 0 0 0 0 0 53

6 16 12 7 0 0 0

| Your exact output will depend on the state of your random number generator.

| We notice that when we just run with 1 random start, the clusters tend to

| group the nonmoving activities together in one cluster. The walking

| activities seem to cluster individually by themselves. You could run the call

| to kmeans with one random start again and you'll probably get a slightly

| different result, but....

| ... instead call kmeans with 3 arguments, the last of which will tell it to

| try more random starts and return the best one. The first 2 arguments should

| be the same as before (sub1 with the last 2 columns removed and centers set

| equal to 6). The third is nstart set equal to 100. Put the result in kClust

| again.

> kClust<-kmeans(sub1[,-c(562,563)],centers=6,nstart=100)

> table(kClust$cluster,sub1$activity)

laying sitting standing walk walkdown walkup

1 18 10 2 0 0 0

2 0 0 0 0 49 0

3 0 37 51 0 0 0

4 3 0 0 0 0 53

5 0 0 0 95 0 0

6 29 0 0 0 0 0

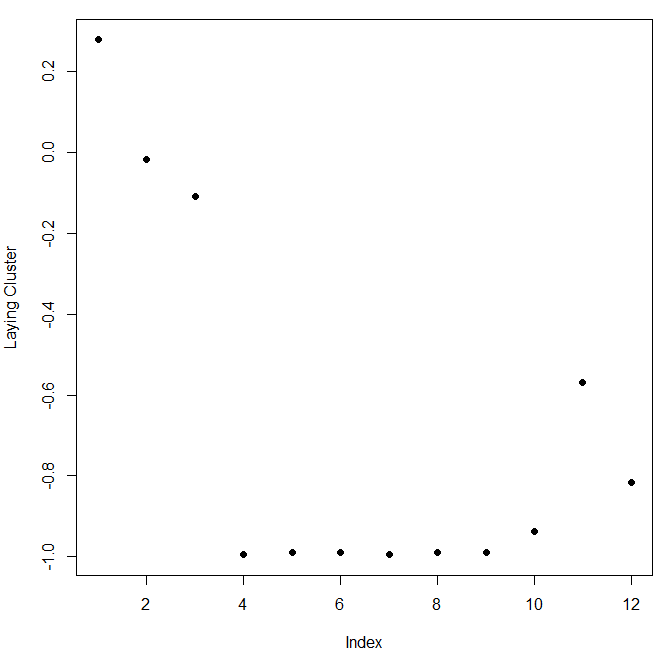
> dim(kClust$centers)

[1] 6 561

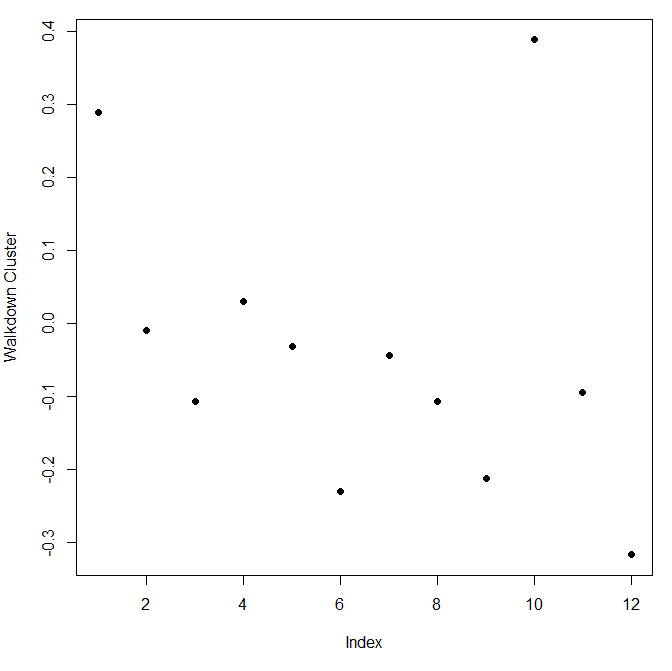
So the centers are a 6 by 561 array. Sometimes it's a good idea to look at the features (columns) of these centers to see if any dominate.

laying<-which(kClust$size==29)

> plot(kClust$centers[laying,1:12],pch=19,ylab="Laying Cluster")



> plot(kClust$centers[walkdown,1:12],pch=19,ylab="Walkdown Cluster")



We'll wrap up here and hope this example convinced you that real world analysis can be frustrating sometimes and not always obvious. You might have to try several techniques of exploratory data analysis before you hit one that pays off and leads you to the questioms that will be the most promising to explore.

We saw here that the sensor measurements were pretty good at discriminating

| between the 3 walking activities, but the passive activities were harder to

| distinguish from one another. These might require more analysis or an

| entirely different set of sensory measurements.

from video:

(swirl) We've read in 2 large zipped files for you using the R command read.table (which is smart enough to unzip the files)

Look at the cluster centers to see what features have moe influence

plot(kClust$center[4,1:10],pch=19,ylab="Cluster Center",xlab="")

pm0<-read.table("RD\_501\_88101\_1999-0.txt",comment.char="#",header=FALSE,sep="l",na.strings="")

cnames<-readLines("RD\_501\_88101\_1999-0.txt",1)

cnames<-strsplit(cnames,"|",fixed=TRUE)

cnames returns a list back, you just want the first value of the list

names(pm0)<-cnames[[1]]

mean(is.na(x0))

[1] 0.1125608

So 11% of the values are missing

**names(pm1)<-make.names(cnames[[1]])**

**##** **"makes syntactically valid names".**

x1<-pm1$Sample>Value

summary(x1)

summary(x0)  
mean(is.na(x1))

[1] 0.05607125

boxplot(x0,x1)

boxplot(log10(x0),…

ngative<-x1<0

sum(negative,na.rm=TRUE)

[1] 26474

date<-pm1$Date

str… integer

dates<-as.Date(as.character(dates),"%Y%m%d")

str("dates")

hist(dates,"month")

hist(dates[negative],"month")

We see the bulk of the negative measurements were taken in the winter months, with a spike in May. Not many of these negative measurements occurred in summer months. We can take a guess that because particulate measures tend to be low in winter and high in summer, coupled with the fact that higher densities are easier to measure, that measurement errors occurred when the values were low. For now we'll attribute these negative measurements to errors. Also, since they account for only 2% of the 2012 data, we'll ignore them.

check monitors in one state:

site0<-unique(subset(pm0,State.Code==36,c(County.Code,Site.ID)))

site1<-unique(subset(pm1,State.Code==36,c(County.Code,Site.ID)))

site0<-paste(site0[,1],site0[,2],sep=".")

site1<-paste(site1[,1],site1[,2],sep=".")

both<-intersect(site0,site1)

pm0$county.site<-with(pm0,paste(County.Code,Site.ID,sep"."))

cnt0<-subset(pm0,State.Code==37&county.site%in%both)

cnt1…

sapply(split(cnt0,cnt0$county.site),nrow)

1.12 1.5 101.3 …

61 122 152 …

county 1 site 12 has 61 observations…

…

pm1sub<-subset(cnt1,County.Code==63&Site.ID==2008)

x1sub<-pm1sub$Sample.Value

par(mfrow=c(1,2),mar=c(4,4,2,1))

plot(dates0,x0sub,pch=20)

abline(h=median(x0sub,na.rm=T))

plot(…x1sub

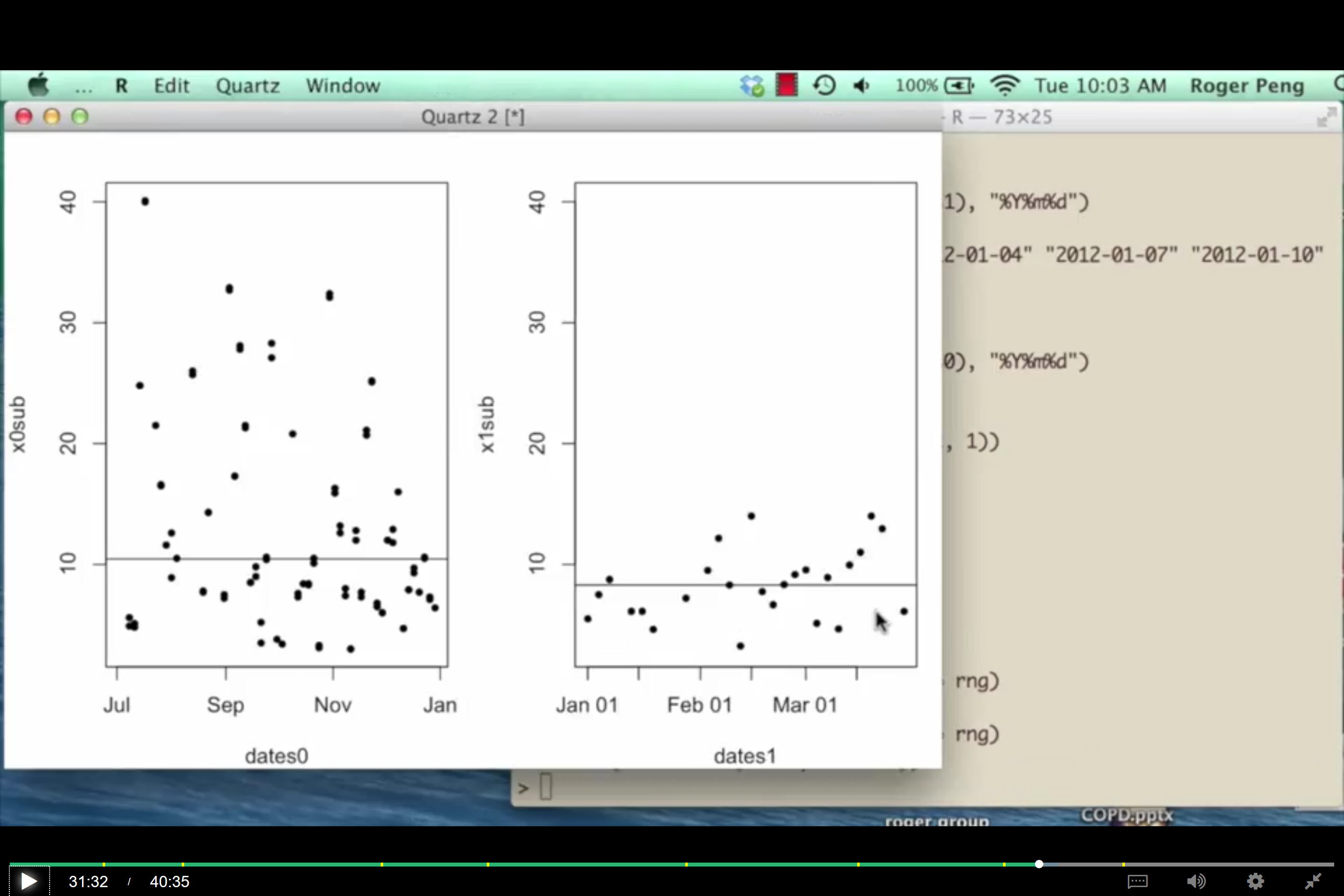
abline(h=median(xisub)…

rng<-range(x0sub,x1sub,na.rm=T)

plot(dates0,x0sub,pch=20,ylim=rng)

…

This shows that not only is there a chronic improvement in air quality, but also there are fewer days with severe pollution.



The last avenue of this data we'll explore (and we'll do it quickly) concerns a comparison of all the states' mean pollution levels. This is important because the states are responsible for implementing the regulations set at the federal level by the EPA.

Create the vector mn0 with a call to the R command with using 2 arguments. The first is pm0. This is the data in which the second argument, an expression, will be evaluated.

mn0<-with(pm0,tapply(Sample.Value,**State.Code,mean**,na.rm=T))

str(mn1)

We see mn0 is a 53 long numerical vector. Why 53 if there are only 50 states? As it happens, pm25 measurements for the District of Columbia (Washington D.C), the Virgin Islands, and Puerto Rico are included in this data. They are coded as 11, 72, and 78 respectively.

str(mn1)

num [1:52(1d)] 10.13 4.75 8.61 10.56 9.28 ...

- attr(\*, "dimnames")=List of 1

..$ : chr [1:52] "1" "2" "4" "5" ...

So mn1 has only 52 entries, rather than 53. We checked. There are no entries for the Virgin Islands in 2012. Call summary now with mn0 as its input.

summary(mn0)

Min. 1st Qu. Median Mean 3rd Qu. Max.

4.862 9.519 12.315 12.406 15.640 19.956

summary(mn1)

Min. 1st Qu. Median Mean 3rd Qu. Max.

4.006 7.355 8.729 8.759 10.613 11.992

We see that in all 6 entries, the 2012 numbers are less than those in 1999. Now we'll create 2 new dataframes containing just the state names and their mean measurements for each year. First, we'll do this for 1999. Create the data frame d0 by calling the function data.frame with 2 arguments. The first is state set equal to names(mn0), and the second is mean set equal to mn0.

**d0<-data.frame(state=names(mn0),mean=mn0)**

d1…

mrg<-merge(d0,d1,by="state")

head(mrg)

state mean.x mean.y

1 1 19.956391 10.126190

2 10 14.492895 11.236059

3 11 15.786507 11.991697

4 12 11.137139 8.239690

5 13 19.943240 11.321364

6 15 4.861821 8.749336

par(mfrow=c(1,1)) ##reset par

Now we'll plot the data to see how the state means changed between the 2 years. First we'll plot the 1999 data in a single column at x=1. The y values for the points will be the state means. Again, we'll use the R command with so we don't have to keep typing mrg as the data environment in which to evaluate the second argument, the call to plot.

For the first column of points, call with with 2 arguments. The first is mrg, and the second is the call to plot with 3 arguments. The first of these is rep(1,52). This tells the plot routine that the x coordinates for all 52 points are 1. The second argument is the second column of mrg or mrg[,2] which holds the 1999 data. The third argument is the range of x values we want, namely xlim set to c(.5,2.5). This works since we'll be plotting 2 columns of points, one at x=1 and the other at x=2.

We see a column of points at x=1 which represent the 1999 state means. For the second column of points, again call with with 2 arguments. As before, the first is mrg. The second, however, is a call to the function points with 2 arguments. We need to do this since we're adding points to an already existing plot. The first argument to points is the set of x values, rep(2,52). The second argument is the set of y values, mrg[,3]. Of course, this is the third column of mrg. (We don't need to specify the range of x values again.)

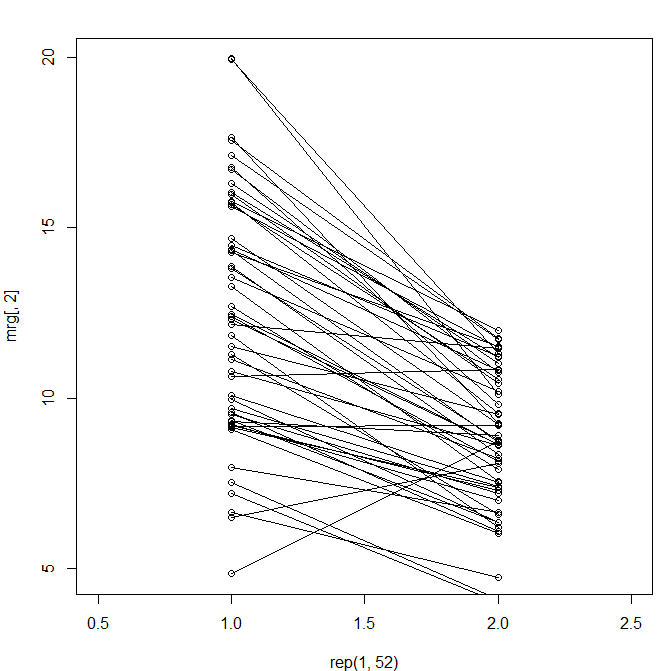
We see a shorter column of points at x=2. Now let's connect the dots. Use the R function segments with 4 arguments. The first 2 are the x and y coordinates of the 1999 points and the last 2 are the x and y coordinates of the 2012 points. As in the previous calls specify the x coordinates with calls to rep and the y coordinates with references to the appropriate columns of mrg.

with(mrg,plot(rep(1999,52),mrg[,2],xlim=c(1998,2013)))

with(mrg,points(rep(2012,52),mrg[,3]))

segments(rep(1999,52),mrg[,2],rep(2012,52),mrg[,3])

We see from the plot that the vast majority of states have indeed improved their particulate matter counts so the general trend is downward. There are a few exceptions. (The topmost point in the 1999 column is actually two points that had very close measurements.)



For fun, let's see which states had higher means in 2012 than in 1999. Just use the mrg[mrg$mean.x < mrg$mean.y, ] notation to find the rows of mrg with this particulate property.

mrg[mrg$mean.x<mrg$mean.y,]

state mean.x mean.y

6 15 4.861821 8.749336

23 31 9.167770 9.207489

27 35 6.511285 8.089755

33 40 10.657617 10.849870

Only 4 states had worse pollution averages, and 2 of these had means that were very close. If you want to see which states (15, 31, 35, and 40) these are, you can check out this website https://www.epa.gov/enviro/state-fips-code-listing to decode the state codes.

<http://stanford.edu/~jgrimmer/RDataManagement.pdf>

25 paginitas

<https://www.stat.berkeley.edu/~spector/Rcourse.pdf>

<https://stackoverflow.com/questions/2628621/how-do-you-use-scoping-assignment-in-r>

<https://www.r-bloggers.com/data-caching/>

<file:///C:/Users/javier.lores/Downloads/v40i01.pdf>

<https://www2.uned.es/pea-metodos-estadisticos-aplicados/>

Graphs

<http://varianceexplained.org/r/teach_ggplot2_to_beginners>

<http://www.ling.upenn.edu/~joseff/rstudy/week4.html>

Sorting data frames plus creating them with col names

<https://www.r-bloggers.com/r-sorting-a-data-frame-by-the-contents-of-a-column/>

<https://stackoverflow.com/questions/43813249/r-round-down-dates-to-first-day-of-the-week>

<http://shop.oreilly.com/product/0636920034407.do>

<http://garrettgman.github.io/tidying/>

Tidy Data <Wickham, H. (2014). Tidy Data. Journal of Statistical Software, 59(10), 1 - 23. doi:[http://dx.doi.org/10.18637/jss.v059.i10>](http://dx.doi.org/10.18637/jss.v059.i10%3E)

<https://stackoverflow.com/questions/1299871/how-to-join-merge-data-frames-inner-outer-left-right>

<https://uc-r.github.io/dplyr>

<https://chartio.com/resources/tutorials/how-to-sort-a-data-frame-by-multiple-columns-in-r/>

<http://vita.had.co.nz/papers/tidy-data.pdf>

Using Census ACS 2012-2016 data, we took the percent of adults in each state that have at least a masters and ranked them from 1-50 with 1 being the most highly educated.

<http://research.collegeboard.org/programs/sat/data/cb-seniors-2013>

On replacing/extracting charactersbefore or after a given one…

<https://stevencarlislewalker.wordpress.com/2013/02/13/remove-or-replace-everything-before-or-after-a-specified-character-in-r-strings/>

<https://stackoverflow.com/questions/15895050/using-gsub-to-extract-character-string-before-white-space-in-r>

<https://www.r-bloggers.com/r-tip-use-istrue/>

<https://blog.exploratory.io/filter-data-with-dplyr-76cf5f1a258e>

dplyr removing na:

<http://kbroman.org/datacarpentry_R_2016-06-01/03-dplyr.html>

Filtering row based on content (grepl):

<https://stackoverflow.com/questions/22850026/filtering-row-which-contains-a-certain-string-using-dplyr>

2011 Journal of Statistical Software paper titled 'Dates and Times Made Easy with lubridate'.

<https://github.com/swirldev/swirl_courses>

<http://data.worldbank.org/data-catalog/GDP-ranking-table>

<http://data.worldbank.org/data-catalog/GDP-ranking-table>

<http://data.worldbank.org/data-catalog/ed-stats>

<http://www.quantmod.com/>  ##for stock historic data

Wearables (x2):

<http://www.insideactivitytracking.com/data-science-activity-tracking-and-the-battle-for-the-worlds-top-sports-brand/>

<http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones>

How to use regular expressions in R:

<https://www.dummies.com/programming/r/how-to-use-regular-expressions-in-r/>

Regular expressions in R:

<http://stat545.com/block022_regular-expression.html>

<https://www.r-bloggers.com/the-complete-catalog-of-argument-variations-of-select-in-dplyr/>

Summarize multiple columns with dplyr:

<https://stackoverflow.com/questions/21644848/summarizing-multiple-columns-with-dplyr>

[Artem Klevtsov](https://stackoverflow.com/users/1863950/artem-klevtsov)

Principles\_of\_Analytic\_Graphs. (Slides for this and other Data Science

| courses may be found at github

| https://github.com/DataScienceSpecialization/courses/. If you care to use

| them, they must be downloaded as a zip file and viewed locally. This lesson

| corresponds to 04\_ExploratoryAnalysis/Principles\_of\_Analytic\_Graphics.)

<https://csgillespie.github.io/efficientR/5-3-importing-data.html>

<https://stackoverflow.com/questions/5182238/replace-x-axis-with-own-values>

scale on y axis:

<https://stackoverflow.com/questions/8918452/r-changing-format-of-scale-on-y-axis>

plotting the rainbow?

<https://gis.stackexchange.com/questions/172779/removing-frame-plot-borders-from-plot-im-output>

accepted

Tufte axes?

Add bty="n" in both plot commands.

For time series, add frame.plot=FALSE for the same effect.

For fancier Tufte axes, see <http://www.cl.cam.ac.uk/~sjm217/projects/graphics/>

plot parameters

<https://www.statmethods.net/advgraphs/parameters.html>

pause execution of program

<https://diego.assencio.com/?index=86c137b502561d44b8be02f06d80ee16>

plotting gridlines

<https://stackoverflow.com/questions/8081931/grid-line-consistent-with-ticks-on-axis>

Slides for swirl:

<https://github.com/DataScienceSpecialization/courses/>

cool things with ggplot:

<http://www.sthda.com/english/wiki/ggplot2-line-plot-quick-start-guide-r-software-and-data-visualization>

Modify axis, label and plot axis ggplot:

<https://ggplot2.tidyverse.org/reference/labs.html>

A guide to customize tickmarks and labels ggplot2:

<http://www.sthda.com/english/wiki/ggplot2-axis-ticks-a-guide-to-customize-tick-marks-and-labels>

Very cool things with ggplot, text, axis, size…

<https://stackoverflow.com/questions/14794599/how-to-change-line-width-in-ggplot>

Heatmaps:

<http://sebastianraschka.com/Articles/heatmaps_in_r.html#clustering>.

on PCA:

<http://arxiv.org/pdf/1404.1100.pdf>.

<http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones>

pollution:

<https://www.health.ny.gov/environmental/indoors/air/pmq_a.htm>

English language

Euclidean distance is distance "as the crow flies".

We can keep going like this in the obvious way and pair up individual points, but as luck would have it, R provides a simple function which you can call which creates a dendrogram for you.

Greatest or largest disctance

We see an interesting display of sorts.

We hope this lesson didn't fluster you or get you too heated!

This picks up, or at least alludes to, the first pattern we added in which affected the last 5 columns of the matrix.

However, closer inspection shows that the entries alternate or bounce up and down as you move from left to right. This hints at the second pattern we added in which affected only even columns of selected rows.

We're starting to see slightly more detail, and maybe if you squint you see a grimacing mouth.

The nonmoving activities still are jumbled together.